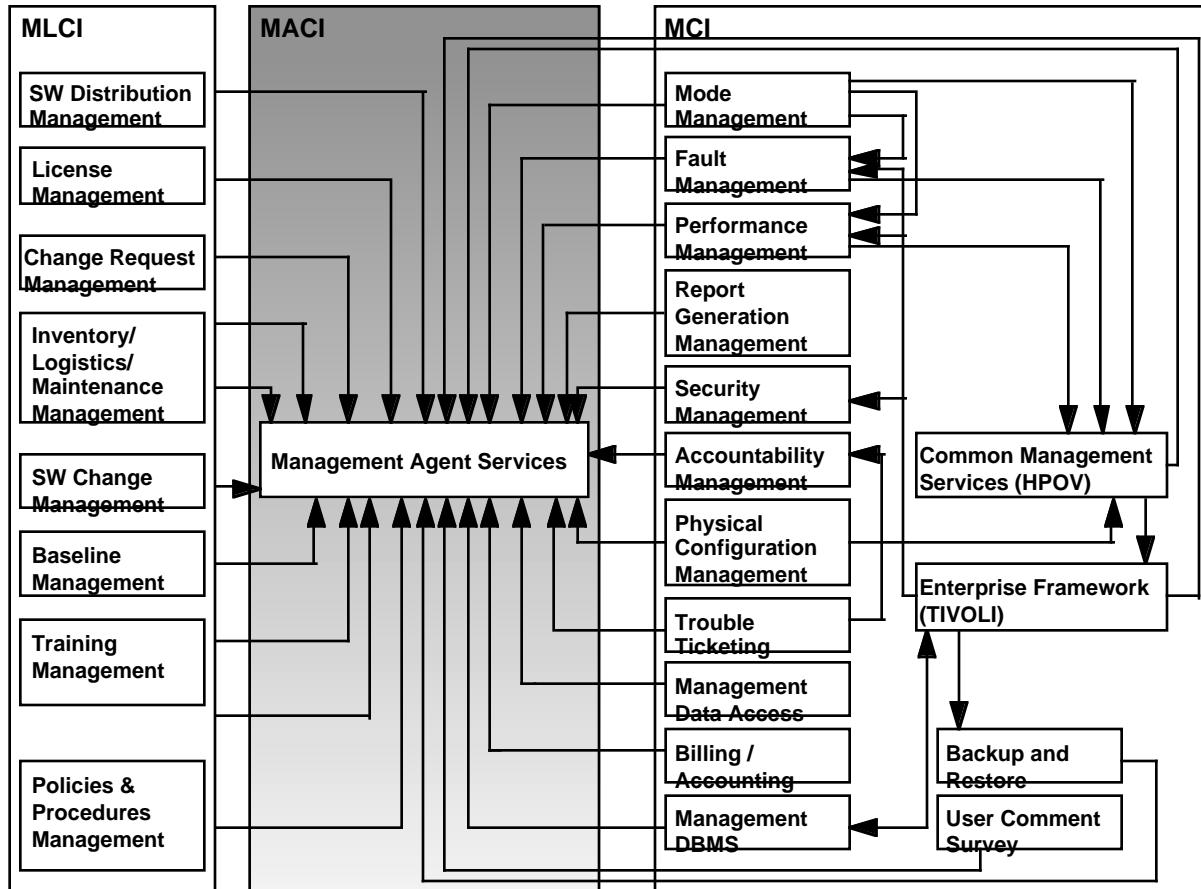


## 4. Management Agent CI

The Management Agent CI consists of the Management Agent Services. The Management Agent Services context is provided in Figure 4-1.



**Figure 4-1. Management Agent CI Context**

### 4.1 Management Agent Services

#### 4.1.1 Management Agent Services Overview

The MSS provides ECS M&O Staff with the capability to manage the ECS enterprise, i.e., to perform network and system management services on all ECS resources, including all SDPS, FOS, and CSMS components.

The enterprise management system is based on the manager-agent model. It consists of management applications, a managed object model, and a management protocol. The management applications reside on managing system(s). They provide the interfaces for the human enterprise

manager to perform management tasks. The managed object model consists of managed objects which are defined to represent the resources being managed. The underlying resources can be physical devices, system software or applications. The management agent resides on each remote host and performs monitoring and control functions for the management applications which are on the managing system(s). The management applications communicate with agents through the management protocol.

The MSS is composed of a variety of management applications providing services such as fault, performance, security, and accountability management for ECS networks, hosts, as well as SDPS and FOS applications. The management applications reside on MSS Server. The management information of remote objects need to be conveyed to the management applications through the Management Agent Service which primarily resides on remote hosts.

The Management Agent Services can be broken down into several distinct parts (see figure 4.1-1) such as the Master agent, the Sentry Agent, the Deputy, the subagent, and the proxy agent. The near real-time Request Tracking capability is detailed in the Accountability section. Refer to Section 6.2 for Request Tracking.

The master agent is responsible for communication of request and responses related to the retrieval of management information. The master agent is provided by a COTS package: PEER Network's Agent. The PEER Agent uses SNMP to communicate with the Network Management System and SMUX to communicate with the subagent.

The Tivoli Sentry, which is the Enterprise Framework agent, is responsible for monitoring operating system level application fault and performance functionality. In addition, it provides a log event adapter which can be configured to examine COTS log files for specific events and then automatically notify Tivoli when certain conditions occur.

The Deputy is responsible for handling secure delivery of requests for setting management information. It is also responsible for generating traps. The Deputy sends the management set requests to the subagent in the form of RPCs. It receives event notifications from the subagent which it converts to traps and forwards them to the server. Deputy will be developed utilizing COTS supplied application programming interface calls to HP OpenView.

The ECS subagent communicates management requests and responses from the master agent and the Deputy to either an ECS developed application or a proxy agent. It supports the MIB extensions and performs local polling on the resources on the host. The subagent is custom developed code that uses some COTS libraries such as PEER tools to make itself remotely manageable.

The proxy agent is provided by the Management Agent Services for the purposes of managing non-SNMP manageable COTS products. Its front-end has the MSS instrumentation code to communicate with the subagent. Its back-end communicates with the COTS. A class library and the basic infrastructure for the Proxy Agent will be provided. Each application will need a single Proxy Agent, and each COTS process will need a corresponding COTS Manager within the Proxy Agent to manage the process.

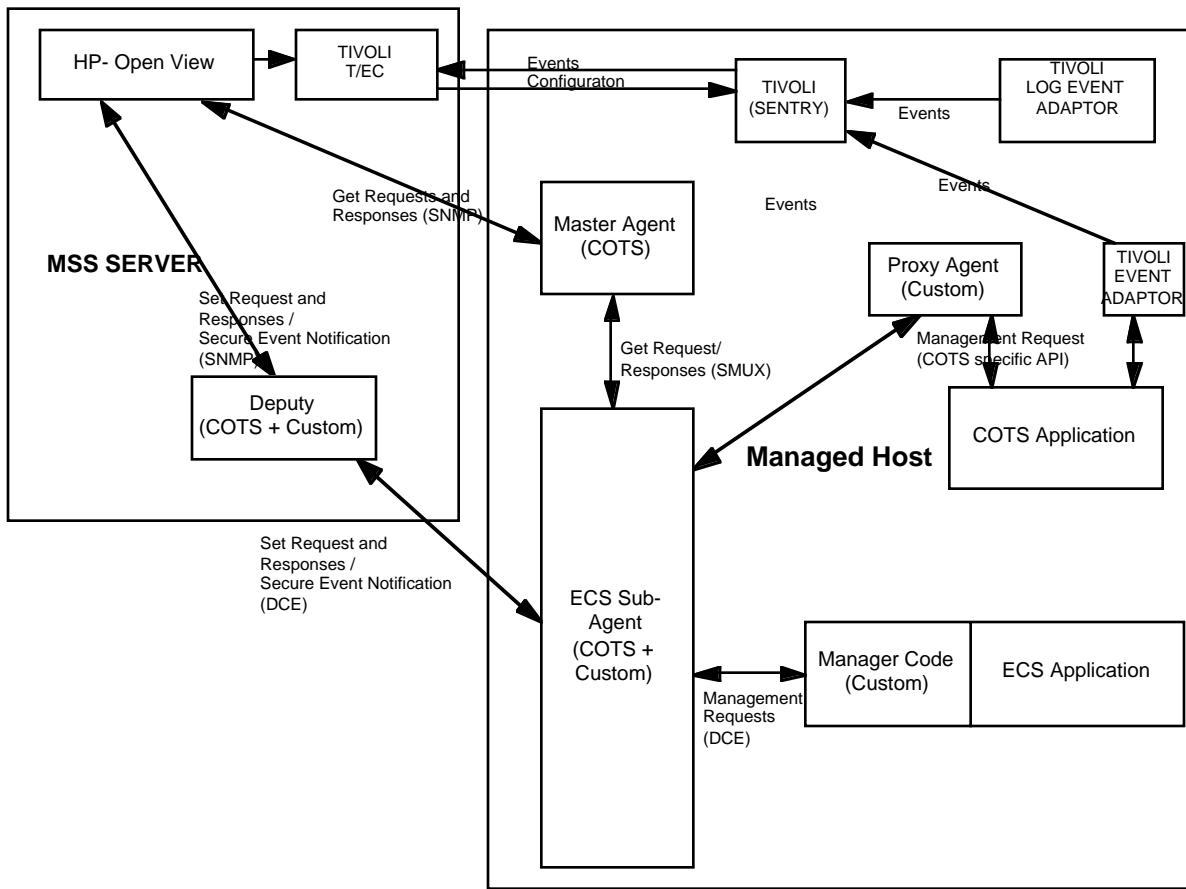
The MSS Management Agent Service provides the following functions:

- Enables the retrieval of ECS managed object values in operational or various test modes.
- Communicates via ECS management protocol with MSS Monitor/Control Service to respond to requests for managed object values.

- Communicates via ECS management protocol with MSS Monitor/Control Service to send ECS management event notifications/traps to the Monitor/Control Service.
- Communicates via ECS management protocol with MSS Monitor/Control Service to receive ECS management Set requests from the Monitor/Control Service.
- Provides an ECS management agent that is configurable to include community, location, log, etc.
- Provides the ECS management agent which can manage network devices.
- Provides the ECS management agent which is extensible for managing host systems.
- Provides the ECS management agent which is extensible for managing ECS applications.
- Provides proxy agent(s) for ECS network devices and applications that cannot be communicated through ECS protocols.
- Performs local polling on hosts to monitor the state of managed ECS resources.
- Provides instrumentation API to application developers to enable the manageability of ECS applications.
- Defines the managed object model to represent the management characteristics of ECS applications. The objects are defined in Management Information Base (MIB).
- Provides Mode management attribute.
- Provides interface to enable mode activation and deactivation.
- Provides an interface to the CSS lifecycle services (start, suspend, resume, shutdown) provided at the process and application level.
- Provides a means to monitor COTS logs and COTS applications for certain predefined conditions and then will send notification to the management console.
- Performs fault and performance level monitoring capabilities.
- Provides an interface between the application and the Distributed Object Framework (DOF) for event processing and distribution.

SNMP has been chosen as the management protocol since it is the defacto and Internet standard protocol for network management in TCP/IP environment. The MSS management applications pass SNMP requests to the agent to retrieve management information. However the setting of management information is done through DCE RPCs for security reasons. The management applications send set requests in the form of RPCs through the Deputy to remote agents.

MSS management applications need to monitor the state of managed resources. It can be done by polling of remote resources. But, remote polling has certain impact on the network traffic. Therefore, the agents perform local polling for the management applications to avoid the costly remote polling and to inform management applications about the state change of managed resources in a more timely manner.



**Figure 4.1-1. Management Agent Services**

MSS requires that on each managed host, standard SNMP MIB II, Host Resource MIB, and the MIBs of network devices are supported by vendor agents. In addition, a managed object model is defined by MSS for ECS applications in SNMP MIB format. The Management Agent Service implements this application MIB. The information contained in the MIB is composed of different types of attributes: configuration, performance, fault, dynamic, static, and traps. Application developers can add their own application specific metrics or parameters to be monitored. However, access methods have to be provided to access these attributes as callback functions.

The Management Agent Services focus on managing the ECS and COTS applications. A set of Instrumentation APIs will be provided to ECS application developers to use for the manageability of ECS applications. ECS applications can be categorized into two general types, OODCE-based or non-OODCE-based applications. Application developers can determine the performance metrics along with their threshold to be monitored. Fault types can also be monitored and counted. For managing non-SNMP resources such as COTS, proxy agents will have to be used, and supplied by the resource provider.

Every application and program needs a configuration file that contains start up information. A file suffix .acfg is expected for the application configuration file and .pcfg for the program

configuration file. This file contains information such as the name of the executable, the command line arguments, and the mode under which the program or application wants to register. Each application or executable resides under a specific mode subdirectory, specified by the mode in the command line argument for the executable.

```
../../../../cfg/ops/myECSapp  
../../../../cfg/ts1/myECStest1  
../../../../cfg/ts2/myECStest2  
...
```

A managed server (EcAgManager) is instantiated for each mode specific instantiation of the application. When the application registers with the subagent, it will specify its mode. When an application is started, subagent's discovery service locates the configuration file based on the mode. The discoverer is responsible for binding with the managed application server, once the managed server notifies that it has started. Once binded, the handle is saved in a binding vector. The discoverer then attempts to retrieve all configuration data from the managed application server and writes them to subagents internal tables. The subagent can provide this application related information to management framework . Once discovered, the application or program appear as icons on the HP Open View map, and may be selected by the operator.

The subagent is also a managed process. However, the subagent itself is mode independent since it must handle applications in various modes. It registers in the mode management hierarchy under the "shared" mode.

After the application has started, it uses the instrumentation APIs provided by the agent to register its metrics. The application developers may inherit from the metrics classes, provided in the instrumentation libraries, to create new classes to gather and report their application specific metric values. The agent allows the application developers to define application specific performance, configuration, and fault metrics that will be monitored in real-time by the subagent.

The performance metrics is any value within the system that helps identify the current performance of the system such as CPU usage, time to process request, etc. Performance metrics may exist at application and process levels. Some of the predefined performance metrics are CPU usage, Memory Usage, Disk I/O, Number of threads (kernel level), and RPC and packet counts. Some of the application specific performance metrics might be average time to process requests or number of unprocessed requests. A performance metrics contains fault, maximum and minimum thresholds, and corresponding fault, maximum and minimum rearm values. The subagent can generate a performance event whenever a performance metric enters a new threshold. It can also change the thresholds and rearm values.

Configuration metrics is a value within application, represented as a string, that can be changed by management framework. They can exist at application and program levels. Some of the predefined configuration metrics are maximum application log file size and application log enabled. Application developers might define their own configuration metrics such as maximum number of RPC entries, or maximum number of simultaneous client connections.

Fault metrics are represented in the form of a counter that provides the number of times a fault has occurred. These metrics can exist at application, program, and process level. There are no

predefined fault metrics. However, some of the application specific fault metrics might be total number of DCE errors, total number of DBMS errors, or total number of operating system errors.

The default configuration metric values and performance thresholds must be specified in the metric configuration file, where they are read by the application at startup. Each time an attribute within the metric is changed, the managed server will update the value within the metric file. The subagent also maintains a copy of these metrics in its internal tables so that the metrics may be monitored as MIB values.

Event handling is provided by Management Agent Service to satisfy the need to dispatch events for orderly and prompt resolution to fix problems. When generating events, applications must provide some of the required fields such as the event category, the event type, event severity, applications CSCI, and its mode. All events are logged locally on each host, to a single file and can be distinguished based on mode using MDA services. A CSS provided logging utility is used for logging events to an application specific file. Event types and subtypes are numeric values that represent the specific type of event that has occurred. Predefined event types are provided by the agent. However, application specific customization of the events is also allowed. Application specific subtypes must be unique within each CSCI. The events are forwarded to the Deputy on the server, which converts them into SNMP traps.

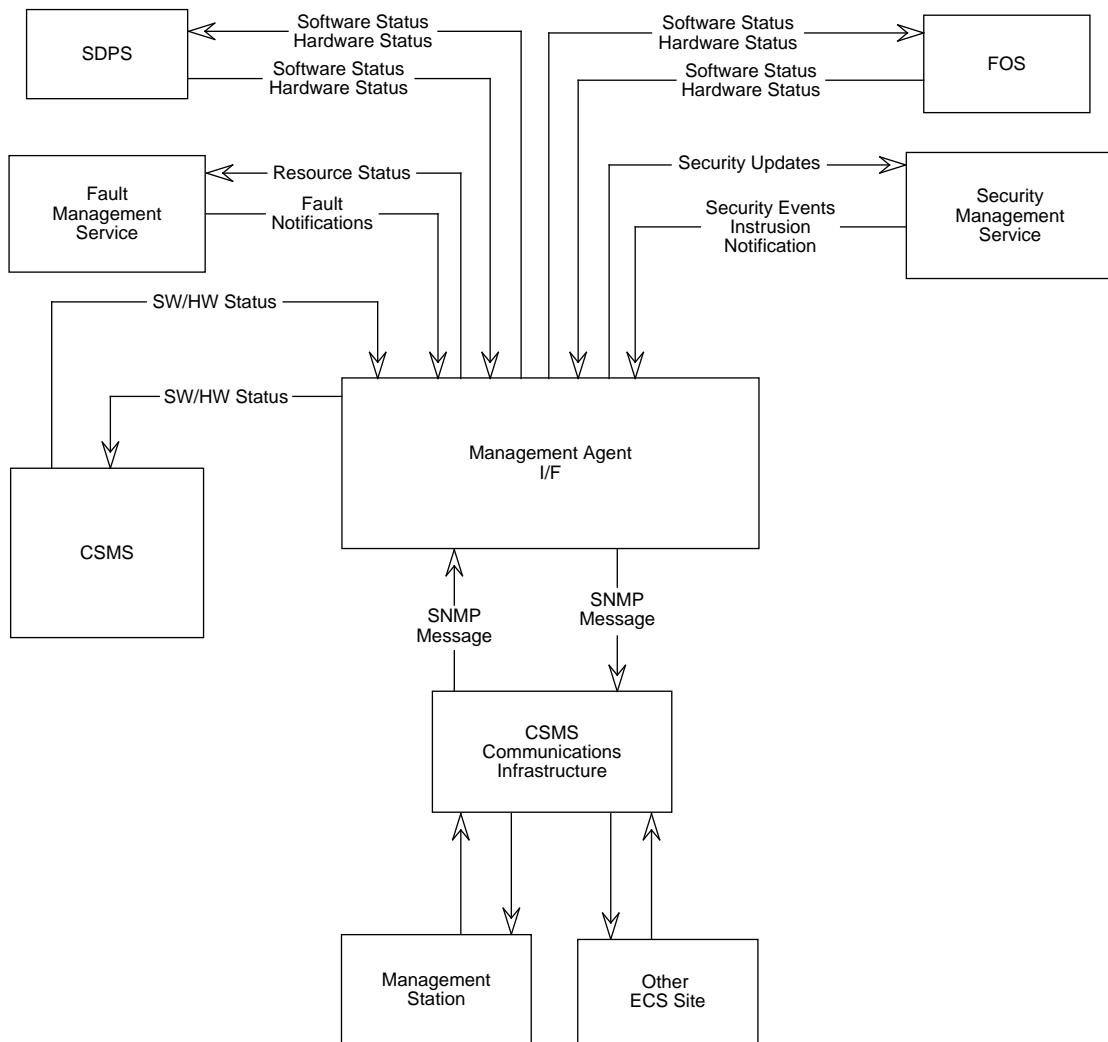
It is assumed that the master agent and the subagent, responsible for the management of applications, are always running. They are started at the boot time of the host. The request to start or to shutdown an application can be issued on the management application. This request will be passed securely to the agent and subagent on remote hosts and the startup or shutdown actions will be performed. The lifecycle services such as startup, suspend, resume, and shutdown, and other requests will trigger event notifications to the management framework. These set requests are bundled within RPCs by the Deputy agent and then sent to the remote subagent's Deputy Gate.

Management applications can make SNMP requests to retrieve management information as MIB values. They can also set certain management information through the use of secure DCE RPC calls. The subagent employs various table entry classes that organize the management information. When a Get request comes from the master agent to retrieve a MIB value, the subagent uses a table manager to locate the appropriate table based on a table ID provided as part of the Get call. It then finds the appropriate row in the table that contains the desired MIB value, based on the index associated with that attribute.

#### **4.1.2 Management Agent Services Context**

The Management Agent Service provides the means to communicate management requests and responses between management applications and managed resources. The managed resources include SDPS, FOS, CSMS applications as well as fault, performance, accountability, security management and Billing and Accounting services.

The management agent services context diagram is shown in Figure 4.1-2.



**Figure 4.1-2. Management Agent Services Context Diagram**

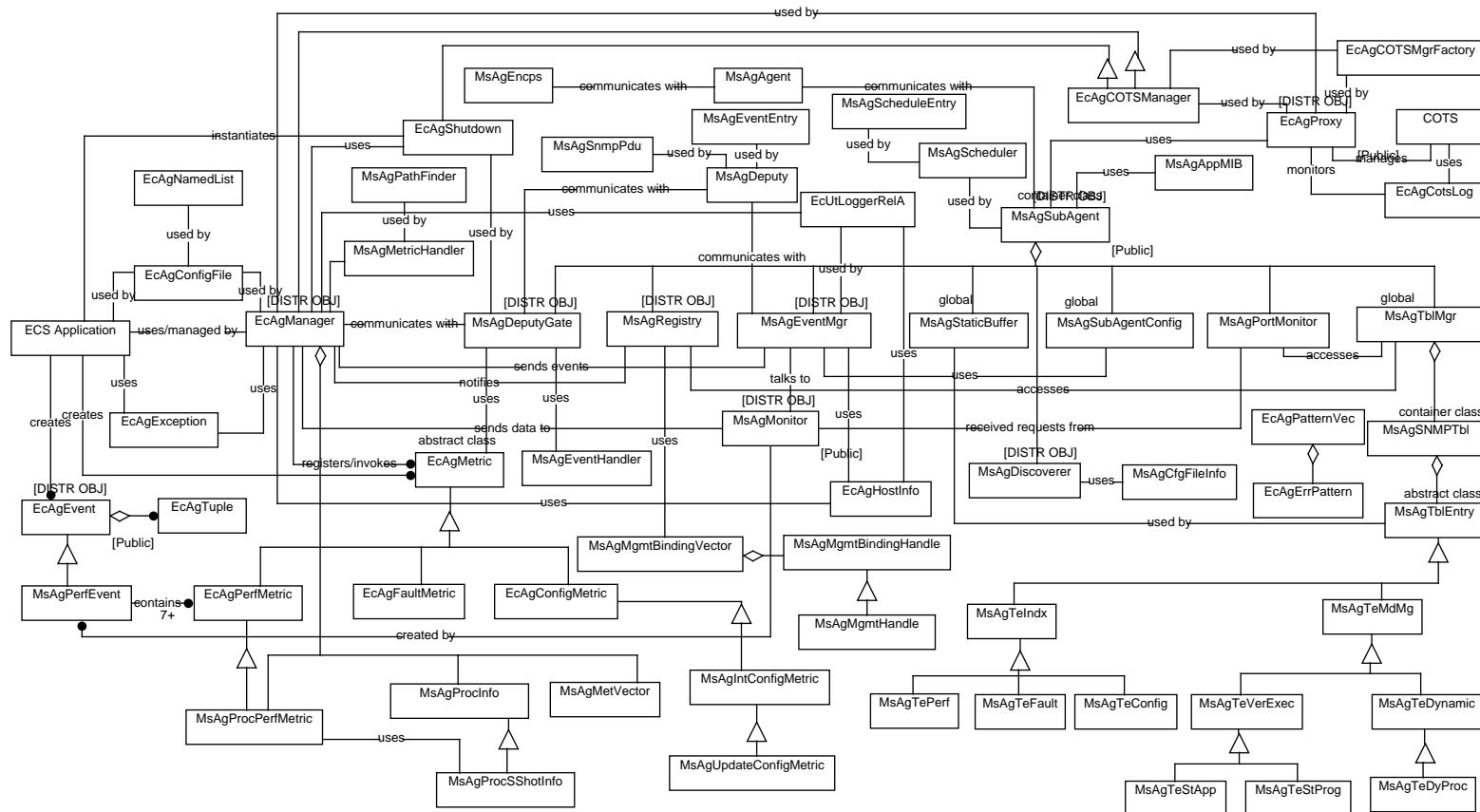
#### 4.1.3 Management Agent Services Object Model

The overview of the Management Agent Services object model is shown in Figure 4.1-3. The different components of the overall model are detailed in Figure 4.1-4 through Figure 4.1-13.

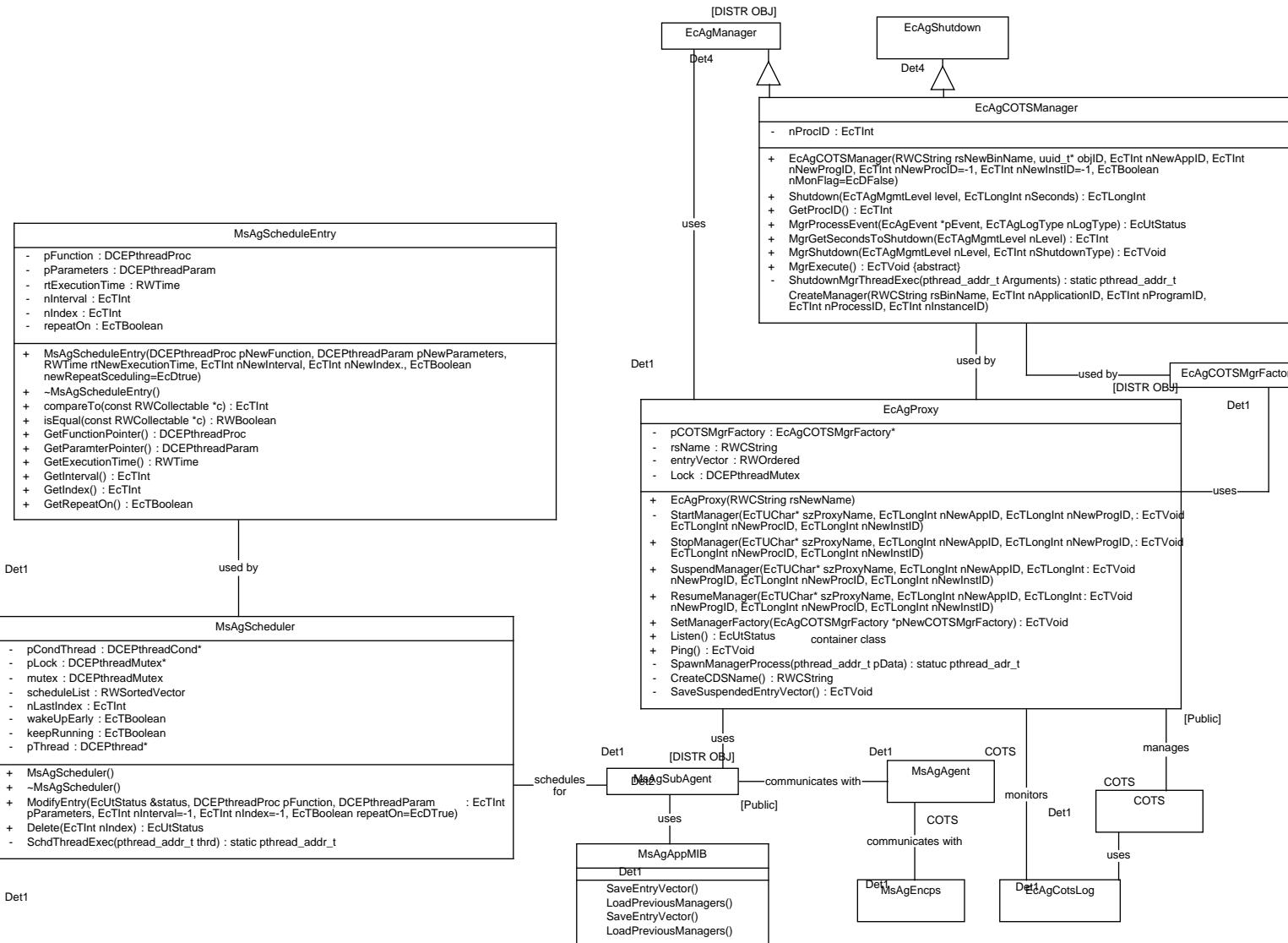
The Management Agent Service object model shows the object classes in the Management Agent Service, the associations among them, their attributes and operations. The classes central to the service are:

- MsAgAgent - This is the extensible SNMP master agent on the host. It receives SNMP requests from management applications through the management framework. It also communicates with subagents to support various MIB extensions. This is a COTS product. Current assumption is Peer Networks' agent and its tool kit to develop subagents.

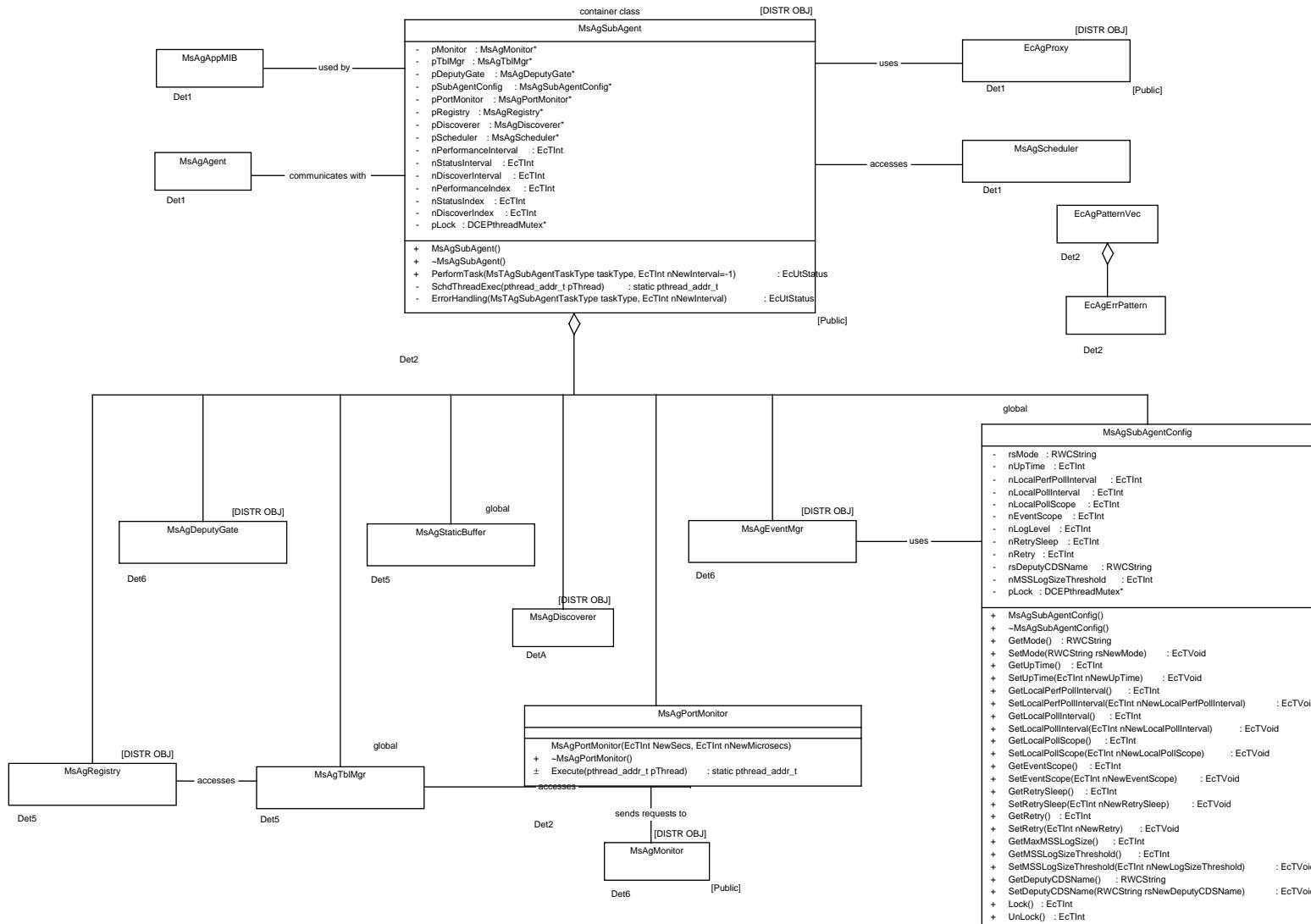
- MsAgSubAgent - This is the subagent primarily for supporting the ECS application MIB. It provides functionality to discover all the installed as well as running applications, programs, and processes periodically as well as on demand. It continually listens for events from various managed objects. It also provides a port monitor that continually listens for requests from the master agent. The Get requests from the master agent are handled by retrieving MIB values through the calling of proper access methods.
- MsAgMonitor - This class is used to perform local monitoring of the state of ECS applications. It checks against the performance threshold and the last state in order to send events if the state has changed.
- EcAgManager - This is part of the instrumentation code. It is used by the subagent or by the monitor to enable the manageability of ECS applications. It provides functionality to enable ECS applications to communicate with the subagent. Through this class the subagent can retrieve application's data and the application can transmit information to the subagent. EcAgManager also provides a method to start the monitoring process of an application. This method is called by the application once it has registered all the metrics that it wants the subagent to manage. The StartMonitoring function informs the subagent that the application is ready to be managed.
- EcAgMetric - This is an abstract class that represents a metric. More specific metrics are inherited from this class such as EcAgPerfMetric, EcAgConfigMetric, and EcAgFaultMetric.
- EcAgEvent - This is part of the instrumentation code. It is used by ECS applications to send out event notifications for logging or eventually trigger the emission of SNMP traps. The event types include fault, performance, security, etc. The severity level includes fatal, error, warning, informational, etc.
- EcAgProxy - This is the proxy agent which enables the ECS subagent to communicate with non-SNMP COTS applications. This proxy agent controls all the COTS manager (EcAgCOTSManager).
- EcAgCOTSManager - This class has the characteristics and functionality of a manager object responsible for managing a single COTS product. It encapsulates all management application functionality. The COTS proxy agent developer is responsible for inheriting from this class and specializing it towards the COTS program to manage.
- MsAgDeputy - This is an entity on MSS Server for management applications to send Set requests to the agent, and to receive secure event notifications from the agent for trap generation. The reason for its existence is to use secure DCE RPC as the transport mechanism as opposed to the use of less reliable SNMP requests between MSS Server and managed hosts.
- MsAgEncps - This is an encapsulator which enables the master agent to communicate with non-Peer SNMP agents.
- MsAgAppMib - This is an SNMP MIB extension for managing applications. It includes the Agent group, the Application group, the Program group, the Process group, and the Traps group. In addition to general attributes, ECS application developers can input their own application-specific metrics or parameters to monitor. But, the access methods have to be provided as callback functions.



**Figure 4.1-3. Management Agent Services Object Model Diagram**



**Figure 4.1-4. Management Agent Services Object Model Detail 1**

**Figure 4.1-5. Management Agent Services Object Model Detail 2**

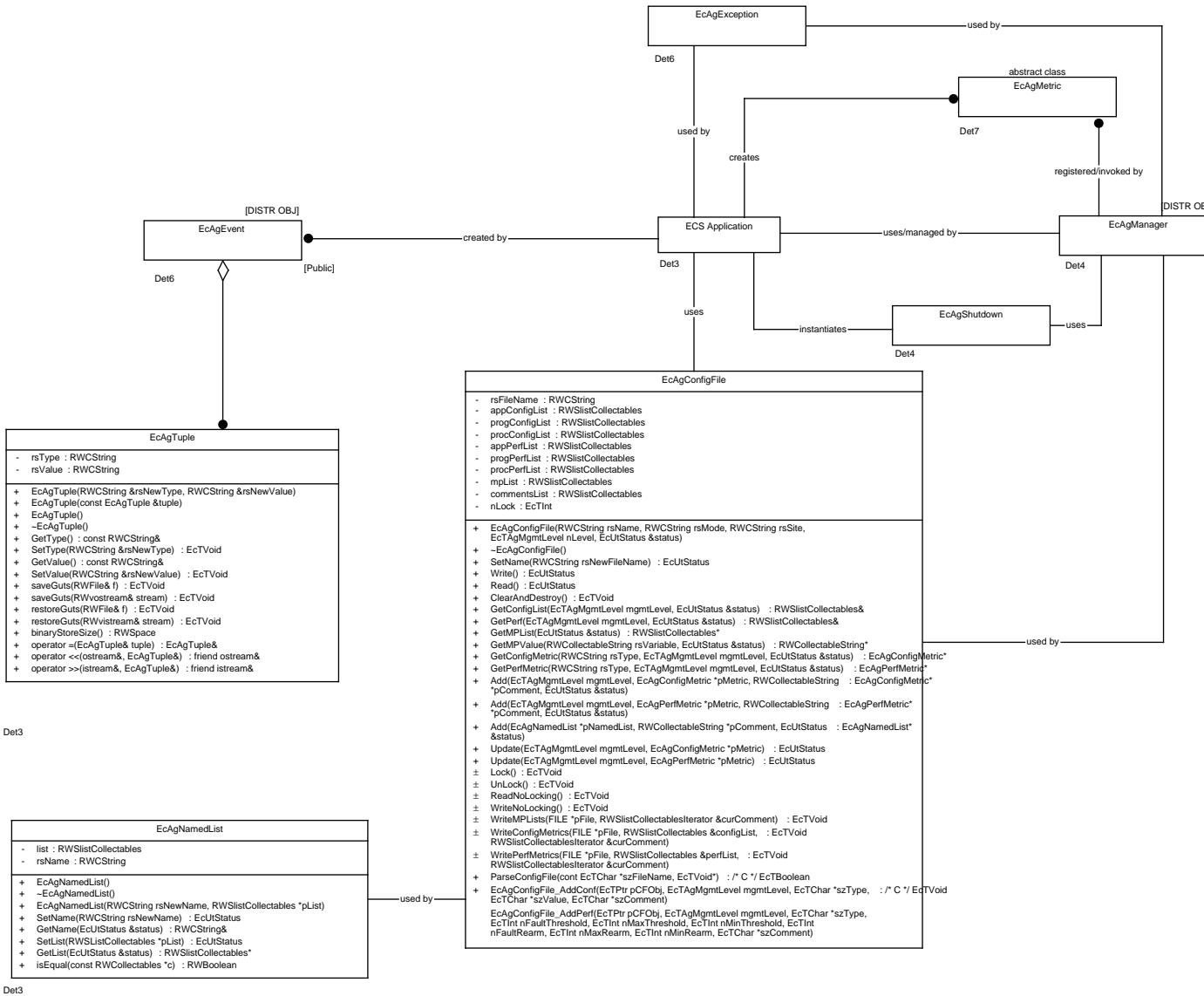
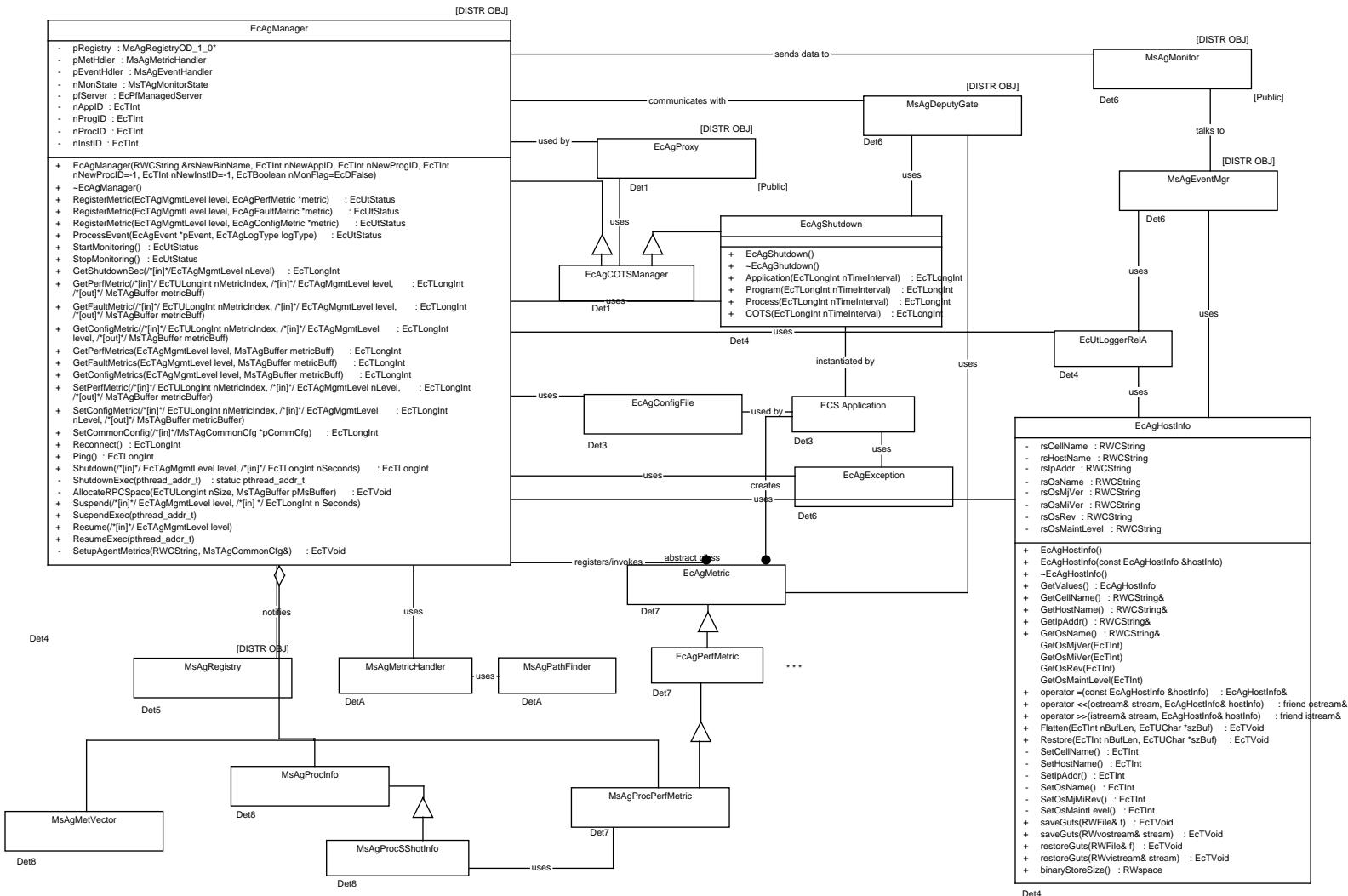


Figure 4.1-6. Management Agent Services Object Model Detail 3



**Figure 4.1-7. Management Agent Services Object Model Detail 4**

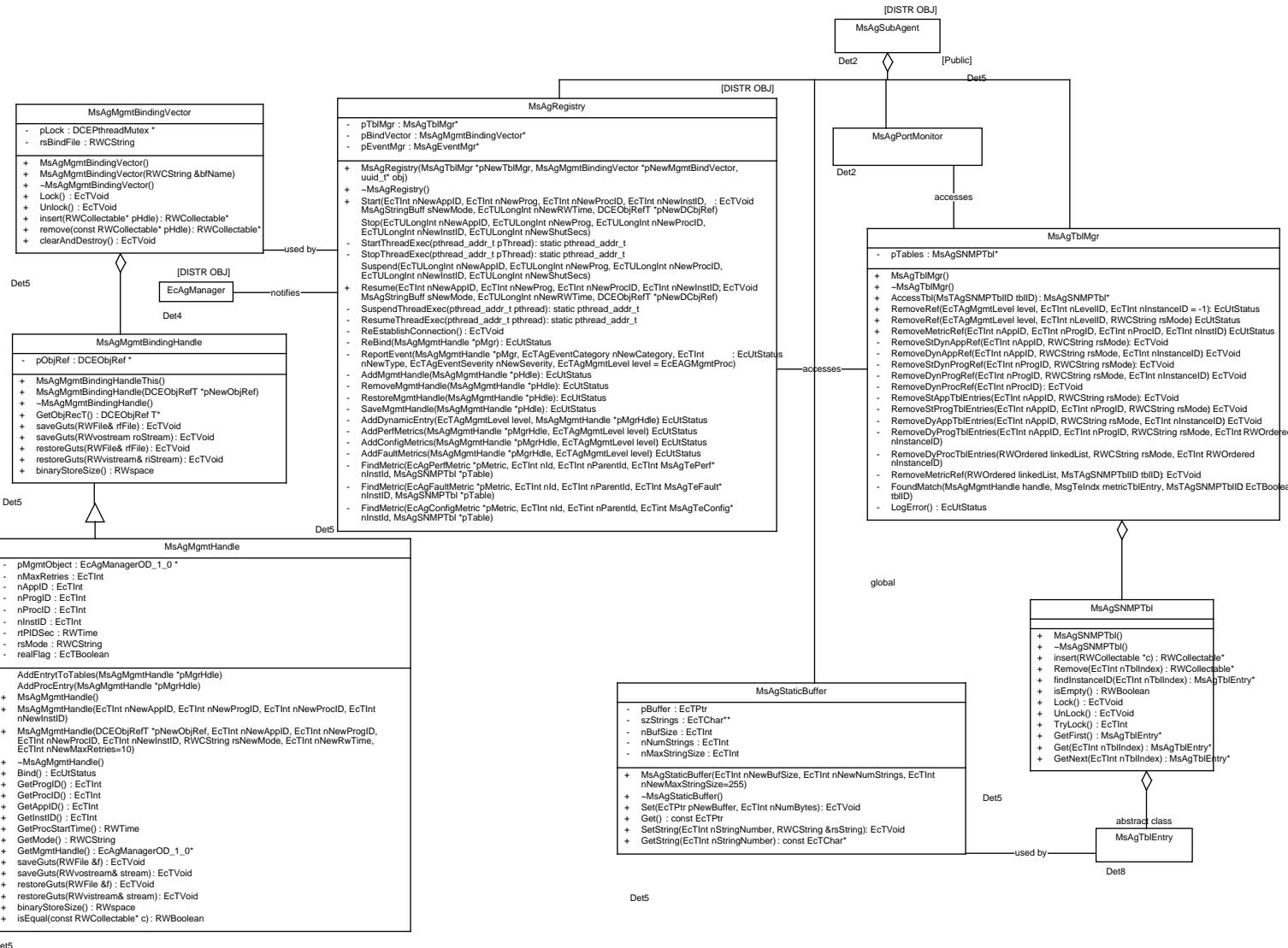
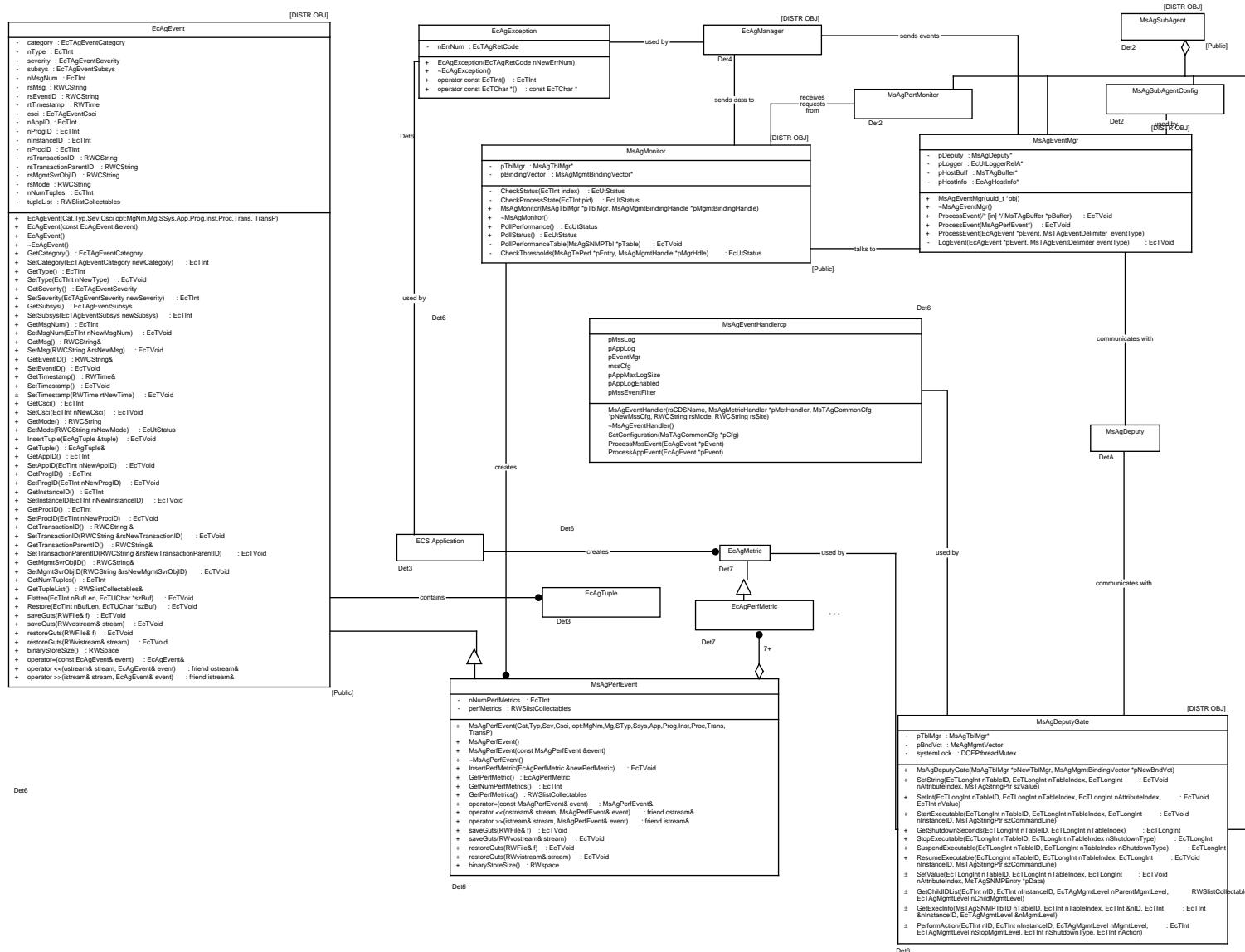
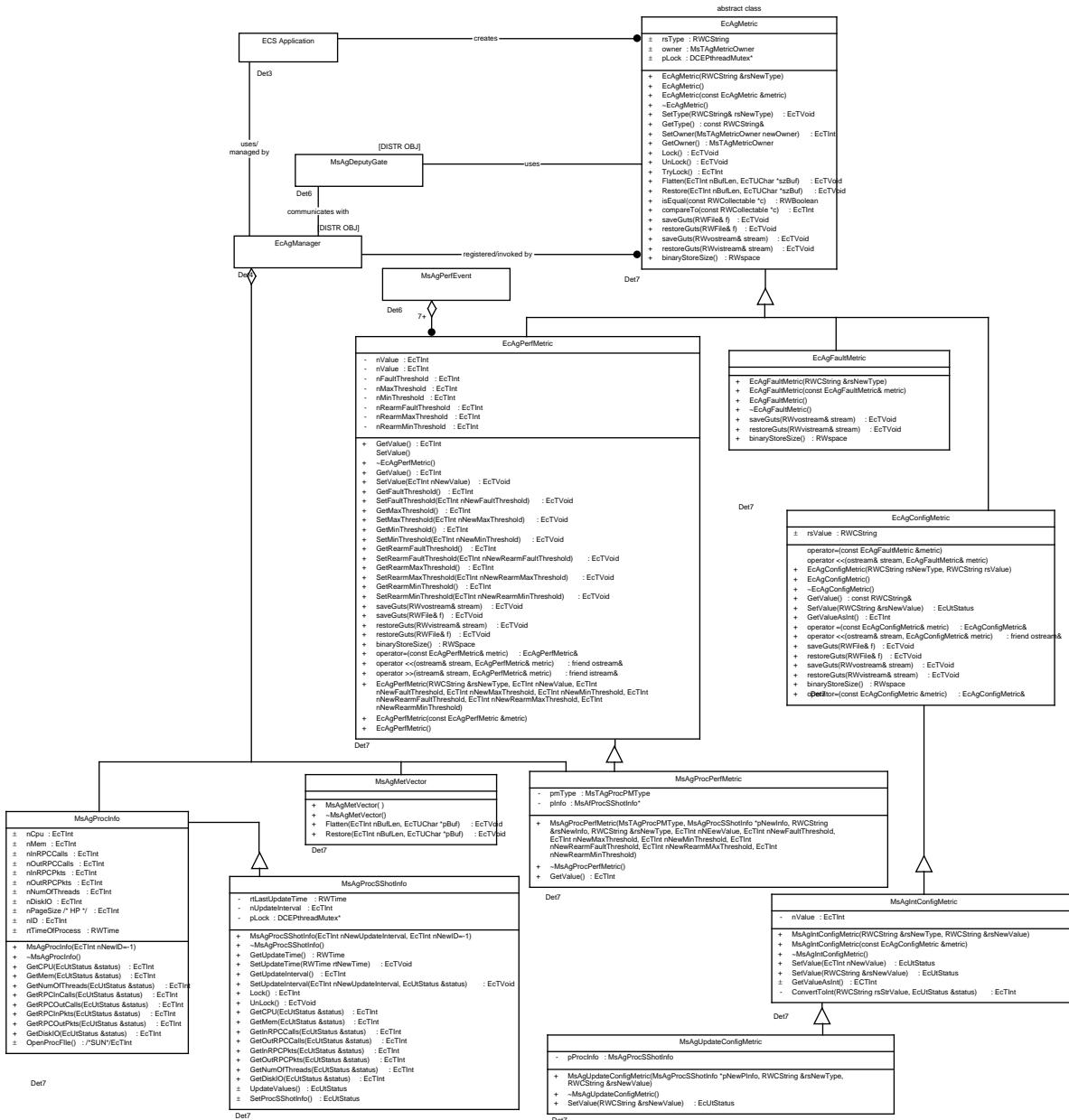


Figure 4.1-8. Management Agent Services Object Model Detail 5



**Figure 4.1-9. Management Agent Services Object Model Detail 6**



**Figure 4.1-10. Management Agent Services Object Model Detail 7**

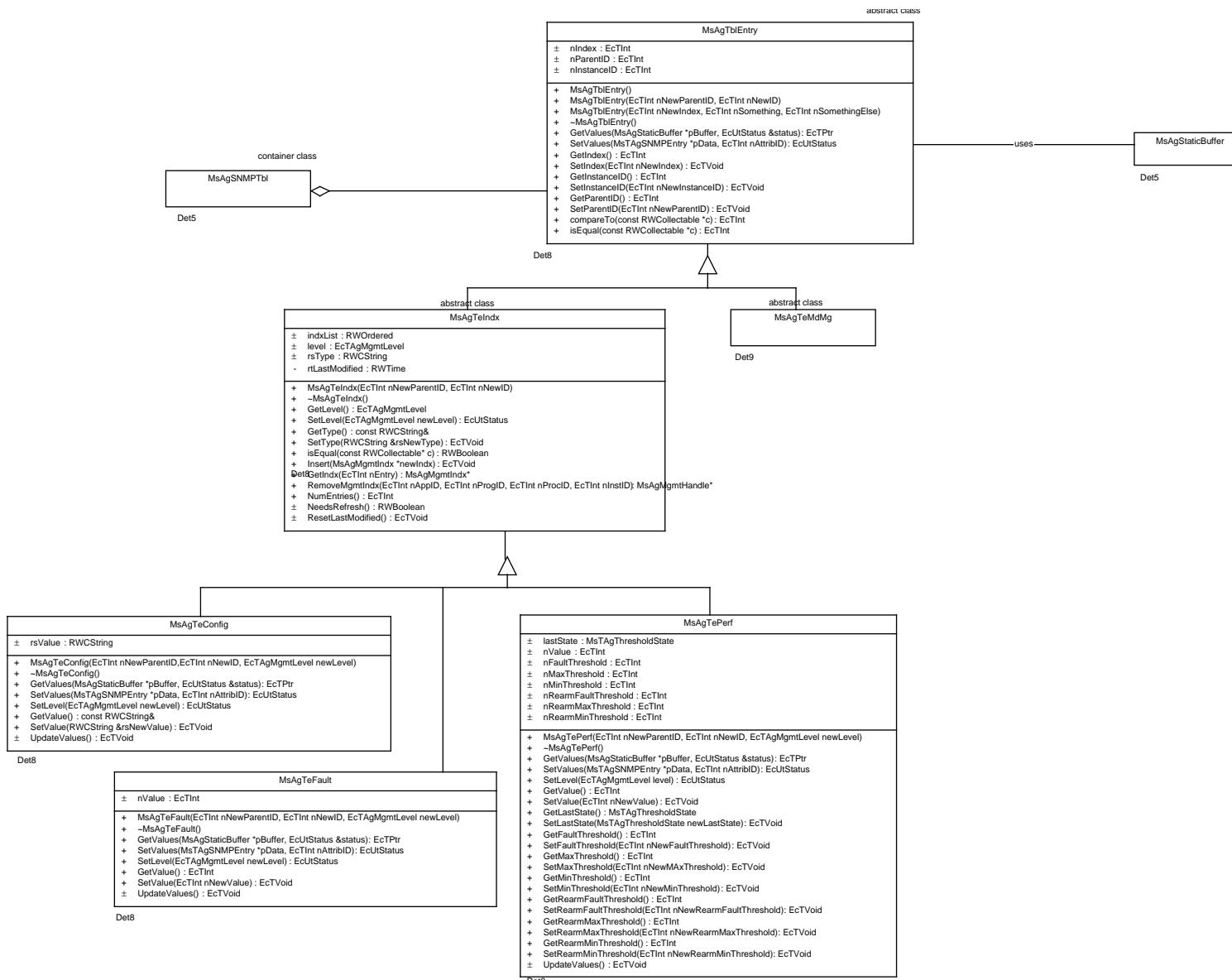
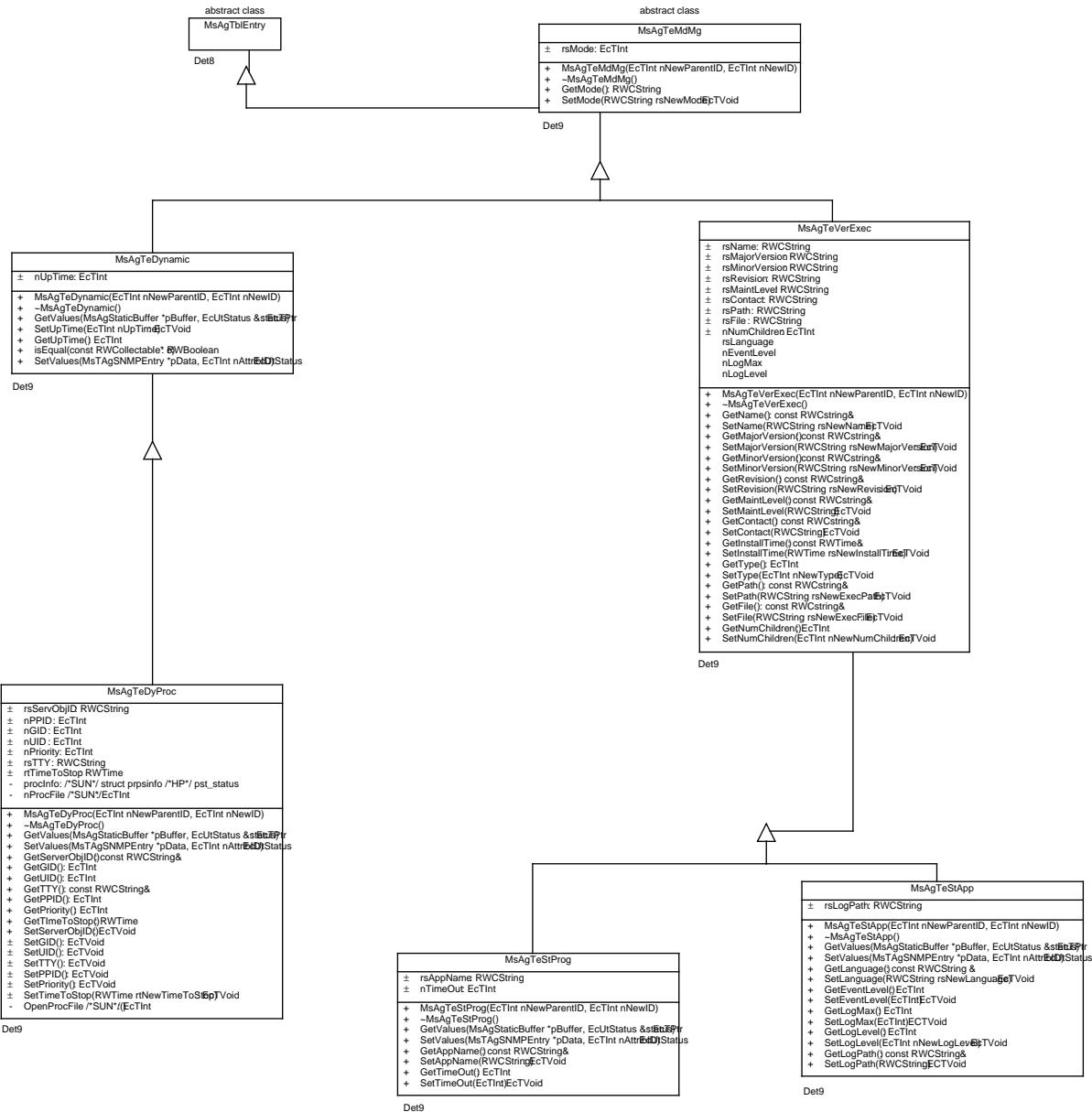


Figure 4.1-11. Management Agent Services Object Model Detail 8



**Figure 4.1-12. Management Agent Services Object Model Detail 9**

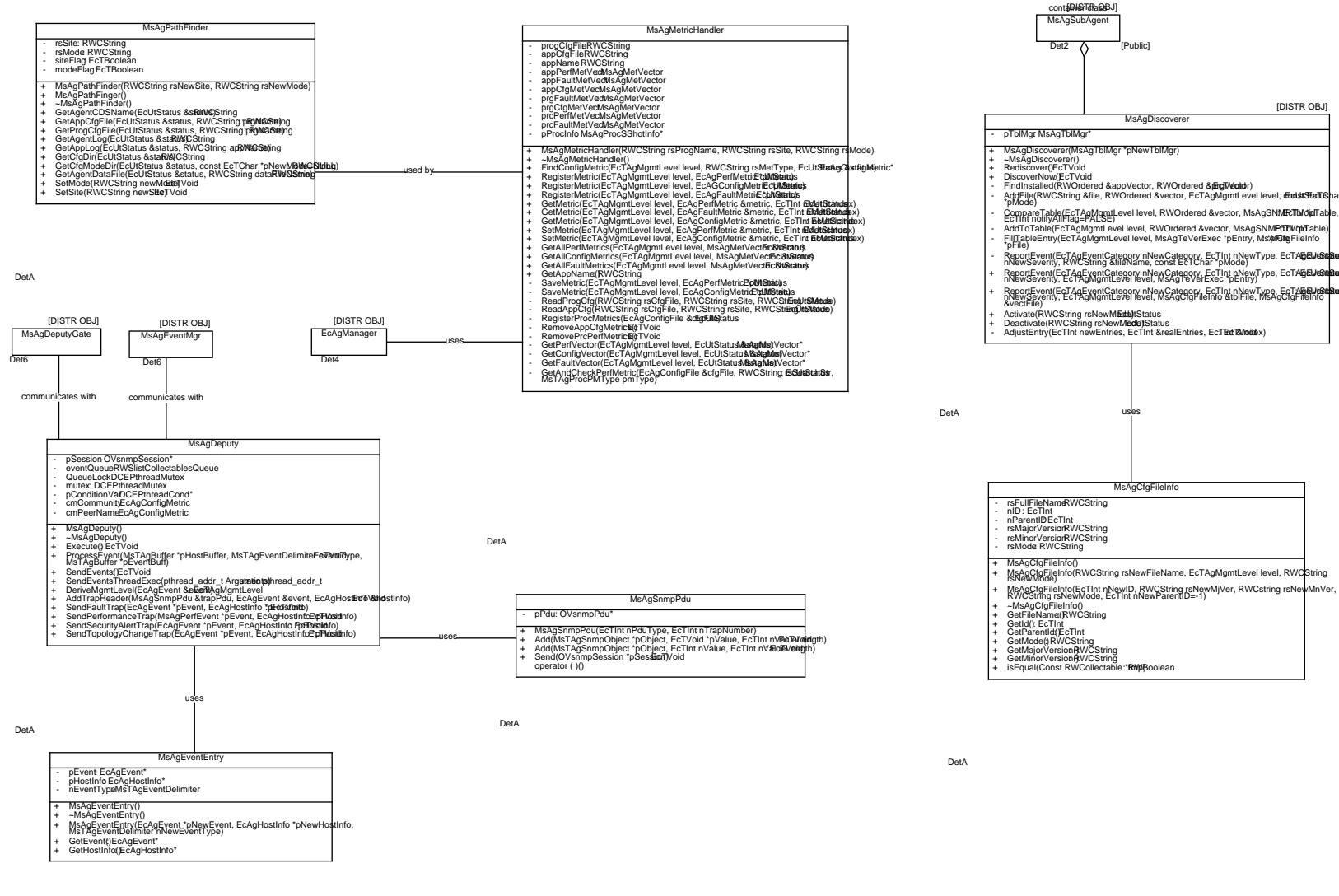
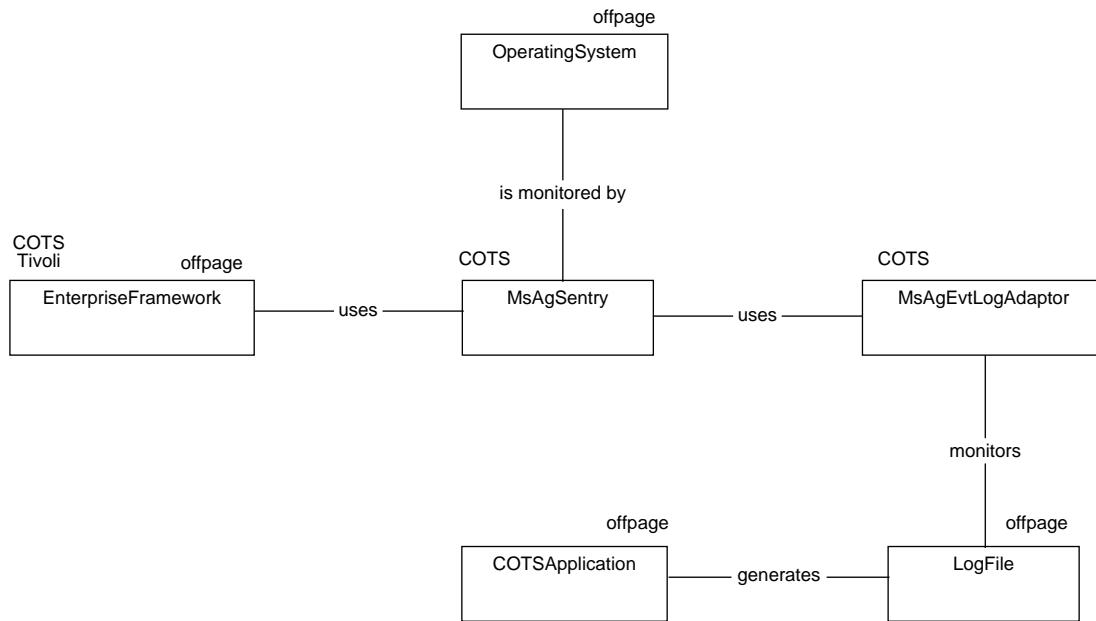


Figure 4.1-13. Management Agent Services Object Model Detail 10



**Figure 4.1-14. Enterprise Framework Agent Service Object Model**

#### 4.1.3.1 COTS Class

Parent Class:Not Applicable

##### Attributes:

None

##### Operations:

None

##### Associations:

The COTS class has associations with the following classes:

- Class: EcAgProxy manages
- Class: EcAgCotsLog uses

#### 4.1.3.2 ECSApplication Class

Parent Class:Not Applicable

**Attributes:**

None

**Operations:**

None

**Associations:**

The ECSApplication class has associations with the following classes:

- Class: EcAgEvent createdby
- Class: EcAgMetric creates
- Class: EcAgShutdown instantiates
- Class: EcAgManager uses/managedby
- Class: EcAgConfigFile uses
- Class: EcAgException uses

#### 4.1.3.3 EcAgCOTSManager Class

Parent Class:EcAgManager

Parent Class:EcAgShutdown

Public:No

Distributed Object:No

Purpose and Description:

This class represents the interface between EcAgManager, EcAgShutdown and the Program specific COTS Manager class.

**Attributes:**

**nProcID** - process ID

**Operations:****EcAgCOTSManager**

Arguments:RWCString rsNewBinName, uuid\_t\* objID, EcTInt nNewAppID, EcTInt nNewProgID, EcTInt nNewProcID=-1, EcTInt nNewInstID=-1, EcTBoolean nMonFlag=EcDFalse

**GetProcID** - returns the process id of the COTS product being managed

Arguments:

**MgrExecute** - virtual method. Classes that inherit from this class wouldn't implement the actual engine of the process proxy agent here.

Arguments:

**MgrGetSecondsToShutdown** - this method, implemented by the application

programmers, must return the total number of seconds required for the application to shutdown at the time the method is called

Arguments:EcTAgMgmtLevel nLevel

**MgrProcessEvent** - process an EcAgEvent

Arguments:EcAgEvent \*pEvent, EcTAgLogType nLogType

**MgrShutdown** - shutdown method that must be reimplemented by the application programmer

Arguments:EcTAgMgmtLevel nLevel, EcTInt nShutdownType

**Shutdown** - shutdown method, executed by EcAgProxyComm, not to be implemented by the application programmers

Arguments:EcTAgMgmtLevel level, EcTLongInt nSeconds

**ShutdownMgrThreadExec** - starting point of the thread spawned by Shutdown. Calls method MgrShutdown

Arguments:pthread\_addr\_t Arguments

### **Associations:**

The EcAgCOTSManager class has associations with the following classes:

Class: EcAgCOTSMgrFactory usedby

Class: EcAgProxy usedby

#### **4.1.3.4 EcAgCOTSMgrFactory Class**

Parent Class:Not Applicable

Public:No

Distributed Object:No

Purpose and Description:

this simple abstract class must be inherited from and reimplemented to create EcAgCOTSManager objects specialized to the particular proxy agent

### **Attributes:**

None

### **Operations:**

**CreateManager** - default constructor. inherited classes would re-implement this method to create different types EcAgCOTSManager objects based on the nProgramID attribute.

Arguments:RWCString rsBinName, EcTInt nApplicationID, EcTInt nProgramID, EcTInt nProcessID, EcTInt nInstanceID

### **Associations:**

The EcAgCOTSMgrFactory class has associations with the following classes:

Class: EcAgCOTSManager usedby

Class: EcAgProxy usedby

### **4.1.3.5 EcAgConfigFile Class**

Parent Class:Not Applicable

Public:No

Distributed Object:No

Purpose and Description:

This class embodies the characteristics and operations of the management configuration file. It provides a mechanism for reading and writing a standard Management configuration file.

### **Attributes:**

**appConfigList** - Linked lists containing all the configuration metrics at the application level.

**appPerfList** - Linked list containing all the performance metrics at the application level.

**commentsList** - linked list of comments

**mpList** - linked list containing all the MP value

**nLock** - File descriptor used for locking and unlocking the configuration file.

**procConfigList** - Linked lists containing all the configuration metrics at the process level.

**procPerfList** - Linked list containing all the performance metrics at the process level.

**progConfigList** - Linked lists containing all the configuration metrics at the program level.

**progPerfList** - Linked list containing all the performance metrics at the program level.

**rsFileName** - File name of the configuration file.

### **Operations:**

**ClearAndDestroy** - Frees all memory allocated for the performance and configuration metrics.

Arguments:

**EcAgConfigFile** - default constructor, accepting the name of the configuration file  
Arguments:RWCString rsName, RWCString rsMode, RWCString rsSite,  
EcTAgMgmtLevel nLevel, EcUtStatus &status

**EcAgConfigFile\_AddConf**

Arguments:EcTPtr pCFObj, EcTAgMgmtLevel mgmtLevel, EcTChar \*szType, EcTChar  
\*szValue, EcTChar \*szComment

**EcAgConfigFile\_AddList**

Arguments:EcTPtr pCFObj, EcTChar \*szName, RWslistCollectables \*pList, EcTChar  
\*szComment

**EcAgConfigFile\_AddPerf**

Arguments:EcTPtr pCFObj, EcTAgMgmtLevel mgmtLevel, EcTChar \*szType, EcTInt  
nFaultThreshold, EcTInt nMaxThreshold, EcTInt nMinThreshold, EcTInt nFaultRarm,  
EcTInt nMaxRarm, EcTInt nMinRarm, EcTChar \*szComment

**GetConfigList**

Arguments:EcTAgMgmtLevel mgmtLevel, EcUtStatus &status

**GetMPList** - return the managed process list

Arguments:EcUtStatus &status

**GetPerf**

Arguments:EcTAgMgmtLevel mgmtLevel, EcUtStatus &status

**Lock** - Locks the configuration file through Unix file locks.

Arguments:

**ParseConfigFile**

Arguments:cont EcTChar \*szFileName, EcTVoid\*

**Read** - Reads the configuration and performance metrics from the configuration file.

Arguments:

**ReadNoLocking** - Reads the config file without locking the config file first.

Arguments:

**SetName** - Sets the name of the configuration file.

Arguments:RWCString rsNewFileName

**UnLock** - UnLocks the configuration file through Unix file locks.

Arguments:

**Update** - Updates the specified performance metrics with the specified values.  
Arguments:EcTAgMgmtLevel mgmtLevel, EcAgPerfMetric \*pMetric

**Write** - Writes all the configuration and performance metrics to the configuration file.  
Arguments:

#### **WriteConfigMetrics**

Arguments:FILE \*pFile, RWStlistCollectables &configList, RWStlistCollectablesIterator &curComment

#### **WriteMPLists**

Arguments:FILE \*pFile, RWStlistCollectablesIterator &curComment

**WriteNoLocking** - Writes the config file without locking the config file first.  
Arguments:

#### **WritePerfMetrics**

Arguments:FILE \*pFile, RWStlistCollectables &perfList, RWStlistCollectablesIterator &curComment

**~EcAgConfigFile** - EcAgConfigFile default destructor doesn't do anything.  
Arguments:

### **Associations:**

The EcAgConfigFile class has associations with the following classes:

Class: EcAgManager usedby  
Class: EcAgNamedList usedby  
Class: ECSApplication uses

#### **4.1.3.6 EcAgConfigMetric Class**

Parent Class:EcAgMetric  
Public:No  
Distributed Object:No  
Purpose and Description:  
This metric contains config data.

### **Attributes:**

**rsValue** - This is an attribute representing a value, i.e. for seconds.

### **Operations:**

**EcAgConfigMetric** - Default constructor required by Rogue Wave.

Arguments:

**EcAgConfigMetric** - This is the default constructor that assigns the type and value.

Arguments:RWCString rsNewType, RWCString rsValue

**GetValue** - This method gets the value.

Arguments:

**GetValueAsInt** - Returns 'value' attribute as integer.

Arguments:

**SetValue**

Arguments:RWCString &rsnewValue

**binaryStoreSize** - This method returns the number of bytes used by the virtual function saveGuts (RWFile&) to store an object.

Arguments:

**operator <<**

Arguments:ostream& stream, EcAgConfigMetric& metric

**operator =**

Arguments:const EcAgConfigMetric& metric

**operator=**

Arguments:const EcAgConfigMetric &metric

**restoreGuts** - This method reads an object's state from an input stream, replacing the previous state.

Arguments:RWvistream& stream

**restoreGuts**

Arguments:RWFile& f

**saveGuts** - This method writes an object's state to an output stream.

Arguments:RWvostream& stream

**saveGuts**

Arguments:RWFile& f

**~EcAgConfigMetric** - This method represents the default destructor.

Arguments:

### **Associations:**

The EcAgConfigMetric class has associations with the following classes:  
None

#### **4.1.3.7 EcAgCotsLog Class**

Parent Class:Not Applicable  
Public:No  
Distributed Object:No  
Purpose and Description:  
This class contains the log of the COTS programs.

### **Attributes:**

None

### **Operations:**

None

### **Associations:**

The EcAgCotsLog class has associations with the following classes:  
Class: EcAgProxy monitors  
Class: COTS uses

#### **4.1.3.8 EcAgErrPattern Class**

Parent Class:Not Applicable

### **Attributes:**

None

### **Operations:**

None

### **Associations:**

The EcAgErrPattern class has associations with the following classes:  
EcAgPatternVec (Aggregation)

#### 4.1.3.9 EcAgEvent Class

Parent Class:Not Applicable

Public:Yes

Distributed Object:Yes

Purpose and Description:

The EcAgEvent defines a distributed object. It provides the capability to dispatch events for orderly and prompt resolution should events occur. The SNMP protocol provides the capability to send traps from agent to SNMP manager. But, the traps are not secure and not reliable. The solution to these problems are using DCE RPC as the transport mechanism for security reasons and sending the traps from MSS Server to the management framework locally. The COTS HP OpenView guarantees the delivery of traps local on one host by using IPC as opposed to UDP. The ECS applications, the EcAgProxy agent, and the MsAgMonitor of the MsAgSubagent can send event notifications to the MsAgSubagent. The MsAgSubagent logs every event into MSS log file. Then, if the severity of the event equals to or is higher than the infoLevel variable, it sends this event notification further to the MsAgDeputy on the MSS Server which in turn convert the event to an SNMP trap and send it locally to the management framework.

##### Attributes:

###### **category**

**csci** - This attribute represents the division within subsystem. i.e.Identifies the CSCI where the event occurred.

**nAppID** - This attribute represents the application ID. i.e. The identifier of the application package

**nInstanceID** - This attribute represents the instanceID. It is either the application's or the program's Instance ID.

**nMsgNum** - This attribute represents the message number (related to message). i.e. Information, Warning, Error, Fatal, Critical

**nNumTuples** - This attribute represents the number of tuples in this event.

**nProcID** - This attribute represents the process ID. i.e. ID of the process which sends out the event

**nProgID** - This attribute represents the program ID. i.e. The identifier of the program

**nType** - This method represents the event type. i.e. System error, Startup, Stop, Process failed, Threshold exceeded, Access attempts

**rsEventID** - This attribute represents the event ID, set by EcAgEvent constructor using

UUID generator. i.e.The UUID of the event

**rsMgmtSvrObjID** - This attribute represents the object ID, server object has a unique identifier, UUID (filled in by EcAgManager) when EcAgManager is created by DCE. i.e. The object ID of the EcAgManager of the application.

**rsMode** - This attribute represents the event mode, i.e. test mode or operational mode. Its string length is at most 8 characters.

**rsMsg** - This attribute represents the message. i.e.The message of the event

**rsTransactionID** - This attribute represents the transaction ID. i.e.UUID of the transaction

**rsTransactionParentID** - This attribute represents the transaction's parent ID. i.e.UUID of the parent transaction

**rtTimestamp**

**severity**

**subsys**

**tupleList**

## Operations:

**EcAgEvent** - The default constructor is only used by RWDECLARE. This method is required by Rogue Wave.

Arguments:

**EcAgEvent** - This method represents the constructor. It uses the values in the argument list to set the corresponding attributes.

Arguments:Cat,Typ,Sev,Csci opt:MgNm,Mg,SSys,App,Prog,Inst,Proc,Trans, TransP

**EcAgEvent** - This method represents the copy constructor.

Arguments:const EcAgEvent &event

**Flatten**

Arguments:EcTInt nBufLen, EcTUChar \*szBuf

**GetAppID** - This method gets the application ID.

Arguments:

**GetCategory** - This method gets the event category.

Arguments:

**GetCsci** - This method gets the division within subsystem.

Arguments:

**GetEventID** - This method gets the event ID, set by Event Manager using UUID generator.

Arguments:

**GetInstanceID** - This method gets the instanceID is either the application's or the program's Instance ID.

Arguments:

**GetMgmtSvrObjID** - This method gets the object ID, server object has a unique identifier, UUID (filled in by Manager) when EcAgManager is created by DCE.

Arguments:

**GetMode** - This method gets the event mode, i.e. test mode or operational mode.

Arguments:

**GetMsg** - This method gets the message.

Arguments:

**GetMsgNum** - This method gets the message number (related to message).

Arguments:

**GetNumTuples** - This method gets the number of tuples in linked list.

Arguments:

**GetProcID** - This method gets the process ID.

Arguments:

**GetProgID** - This method gets the program ID.

Arguments:

**GetSeverity**

Arguments:

**GetSubsys** - This method gets the subsystem of person creating event.

Arguments:

**GetTimestamp** - This method gets the event timestamp.

Arguments:

**GetTransactionID** - This method gets the transaction ID.

Arguments:

**GetTransactionParentID** - This method gets the transaction's parent ID.

Arguments:

**GetTuple** - This method gets a tuple into the linked list of tuples. It will call Rogue Wave's 'get' function to retrieve an element from the rogue wave linked list.

Arguments:

**GetTupleList** - This method gets a list of tuples.

Arguments:

**GetType** - This method gets the event type.

Arguments:

**InsertTuple**

Arguments:EcAgTuple &tuple

**Restore**

Arguments:EcTInt nBufLen, EcTUChar \*szBuf

**SetAppID** - This method sets the application ID.

Arguments:EcTInt nNewAppID

**SetCategory**

Arguments:EcTAgEventCategory newCategory

**SetCsci** - This method sets the division within subsystem.

Arguments:EcTInt nNewCsci

**SetEventID** - This method sets the event ID, set by Event Manager using UUID generator.

Arguments:

**SetInstanceID** - This method sets the instance ID.

Arguments:EcTInt nNewInstanceID

**SetMgmtSvrObjID**

Arguments:RWCString &rsNewMgmtSvrObjID

**SetMode**

Arguments:RWCString rsNewMode

**SetMsg**

Arguments:RWCString &rsNewMsg

**SetMsgNum** - This method sets the message number (related to message).  
Arguments:EcTInt nNewMsgNum

**SetProcID** - This method sets the process ID.  
Arguments:EcTInt nNewProcID

**SetProgID** - This method sets the program ID.  
Arguments:EcTInt nNewProgID

**SetSeverity**  
Arguments:EcTAgEventSeverity newSeverity

**SetSubsys**  
Arguments:EcTAgEventSubsys newSubsys

**SetTimestamp** - This method sets the event timestamp.  
Arguments:

**SetTimestamp**  
Arguments:RWTime rtNewTime

**SetTransactionID**  
Arguments:RWCString &rsNewTransactionID

**SetTransactionParentID**  
Arguments:RWCString &rsNewTransactionParentID

**SetType** - This method sets the event type.  
Arguments:EcTInt nNewType

**binaryStoreSize** - This method returns the number of bytes used by the virtual function saveGuts (RWFile&) to store an object.  
Arguments:

**operator <<**  
Arguments:ostream& stream, EcAgEvent& event

**operator >>**  
Arguments:istream& stream, EcAgEvent& event

**operator=** - This method copies an event.  
Arguments:const EcAgEvent& event

**restoreGuts** - This method reads an object's state from an binary file, using RWFile,

replacing the previous state.

Arguments:RWFile& f

**restoreGuts** - This method reads an object's state from an input stream, replacing the previous state.

Arguments:RWvistream& stream

**saveGuts** - This method writes an object's state to a binary file, using class RWFile.

Arguments:RWFile& f

**saveGuts** - This method writes an object's state to an output stream. virtual EcTVoid

saveGuts ( RWvostream& ) const {};

Arguments:RWvostream& stream

**-EcAgEvent** - This method specifies the destructor of the class.

Arguments:

### Associations:

The EcAgEvent class has associations with the following classes:

Class: ECSApplication createdby

#### 4.1.3.10 EcAgException Class

Parent Class:Not Applicable

Public:No

Distributed Object:No

Purpose and Description:

This class is used by EcAg classes to throw exceptions.

### Attributes:

#### nErrNum

### Operations:

#### **EcAgException**

Arguments:EctAgRetCode nNewErrNum

**operator const EcTChar \*** - This method can typecast this class to get the verbose form of the error.

Arguments:

**operator const EcTInt** - typecast error to integer.

Arguments:

**~EcAgException** - Default destructor.

Arguments:

### **Associations:**

The EcAgException class has associations with the following classes:

Class: EcAgManager usedby

Class: ECSApplication uses

#### **4.1.3.11 EcAgFaultMetric Class**

Parent Class:EcAgMetric

Public:No

Distributed Object:No

Purpose and Description:

This metric contains fault data.

### **Attributes:**

**nValue** - This attribute represents a counter of fault types.

### **Operations:**

**EcAgFaultMetric** - The default constructor is only used by RWDECLARE. This method is required by Rogue Wave.

Arguments:

**EcAgFaultMetric** - This method represents the constructor which sets the type and initializes the value.

Arguments:RWCString &rsNewType

**EcAgFaultMetric** - This method represents the copy constructor.

Arguments:const EcAgFaultMetric& metric

**GetValue** - This method gets the value.

Arguments:

**SetValue** - This method sets the value (i.e. increments it).

Arguments:

**binaryStoreSize** - This method returns the number of bytes used by the virtual function saveGuts (RWFile&) to store an object.

Arguments:

**operator <<**

Arguments:ostream& stream, EcAgFaultMetric& metric

**operator=**

Arguments:const EcAgFaultMetric &metric

**restoreGuts** - This method reads an object's state from an input stream, replacing the previous state.

Arguments:RWvistream& stream

**saveGuts** - This method writes an object's state to an output stream.

Arguments:RWvostream& stream

**~EcAgFaultMetric** - This method represents the default destructor.

Arguments:

### **Associations:**

The EcAgFaultMetric class has associations with the following classes:

None

#### **4.1.3.12 EcAgHostInfo Class**

Parent Class:Not Applicable

Public:No

Distributed Object:No

Purpose and Description:

This class provides the additional information associated with an event, needed by CSS logger and EcAgManager, MsAgEventMgr.

### **Attributes:**

**rsCellName** - This attribute represents the cell name.

**rsHostName** - This attribute represents the host name.

**rsIpAddr** - This attribute represents the IP address.

**rsOsMaintLevel** - This attribute represents the OS maintenance level.

**rsOsMiVer** - This attribute represents the OS minor version.

**rsOsMjVer** - This attribute represents the OS major version.

**rsOsName** - This attribute represents the OS name.

**rsOsRev** - This attribute represents the OS revision level.

### Operations:

**EcAgHostInfo** - This method represents the default constructor.

Arguments:

**EcAgHostInfo** - copy constructor

Arguments:const EcAgHostInfo &hostInfo

#### Flatten

Arguments:EcTInt nBufLen, EcTUChar \*szBuf

**GetCellName** - This method gets the cell name.

Arguments:

**GetHostName** - This method gets the host name.

Arguments:

**GetIpAddr** - This method gets the IP address.

Arguments:

**GetOsMaintLevel** - This method gets the OS maintenance level.

Arguments:EcTInt

**GetOsMiVer** - This method gets the OS minor version.

Arguments:EcTInt

**GetOsMjVer** - This method gets the OS major version.

Arguments:EcTInt

**GetOsName** - This method gets the OS name.

Arguments:

**GetOsRev** - This method gets the OS revision level.

Arguments:EcTInt

**GetValues** - This method gets all values.

Arguments:

**Restore**

Arguments:EcTInt nBufLen, EcTUChar \*szBuf

**SetCellName** - This method sets the DCE cell name attribute.

Arguments:

**SetHostName** - This method sets the host name attribute.

Arguments:

**SetIpAddr** - This method sets the ip address attribute.

Arguments:

**SetOsMaintLevel** - This method sets the os maintenance level attribute.

Arguments:

**SetOsMjMiRev** - This method sets the os major/minor version & revision level attributes.

Arguments:

**SetOsName** - This method sets the os name attribute.

Arguments:

**binaryStoreSize** - Return the number of bytes used by the virtual function saveGuts (RWFile&) to store an object.

Arguments:

**operator <<**

Arguments:ostream& stream, EcAgHostInfo& hostInfo

**operator =**

Arguments:const EcAgHostInfo &hostInfo

**operator >>**

Arguments:istream& stream, EcAgHostInfo& hostInfo

**restoreGuts** - Read an object's state from an binary file, using RWFile, replacing the previous state.

Arguments:RWFile& f

**restoreGuts** - Read an object's state from an input stream, replacing the previous state.

Arguments:RWvistream& stream

**saveGuts** - Write an object's state to a binary file, using class RWFile.

Arguments:RWFile& f

**saveGuts** - Write an object's state to an output stream.

Arguments:RWvostream& stream

**~EcAgHostInfo** - This method represents the default destructor.

Arguments:

#### **Associations:**

The EcAgHostInfo class has associations with the following classes:

Class: MsAgEventMgr usedby  
Class: EcAgManager uses  
Class: EcUtLoggerRelA uses

#### **4.1.3.13 EcAgManager Class**

Parent Class:Not Applicable

Public:Yes

Distributed Object:Yes

Purpose and Description:

This class defines a distributed object. It incorporates most of the manager control and instrumentation functionality for ECS applications and COTS. ECS application developers have to use the class library to instrument their code and to provide callback routines so that the ECS applications are manageable by MSS. The EcAgProxy developer has to do the same. The Application MIB provides table format for Fault, Performance, and Configuration parameters to be added by application developers. In order to monitor these parameters, application developers have to provide callback routines for the Subagent to retrieve the value of these parameters. This object provides the capability for the developers to register all the callback functions required. The callback mechanism provided by this object is a generic one. Application developer has to register the callback function first. When the Get request comes in, it will trigger the execution of the callback function to retrieve the value. This object can be invoked by the MsAgDeputy to send Set requests to the MsAgSubagent. The MsAgSubagent can invoke the methods of this object to send Get, Set, or other requests to ECS applications or to the Proxy of COTS. The data structures that application developers have to use are based on the Application MIB definition. The following are examples:

```
Action Type Data Structure
typedef struct { char PerfType[n+1]; int PerfValue; int PerfThreshold; }

Get_Fault
typedef struct { char FaultType[n+1]; int FaultValue; }

Get_Config
typedef struct { char CfgParam[n+1]; char value[m+1]; }

Shutdown
typedef struct { int now; }
```

#### **Attributes:**

**nAppID** - The subagent keeps track of which processes belong to a particular program and which programs belong to a particular application. The nAppID, nProgID, and nProcID are used for this purpose. The nInstID distinguishes multiple instances of the same program of the same application.

Data Type:EcTInt

Privilege:Private

Default Value:

**nInstID** - The subagent keeps track of which processes belong to a particular program and which programs belong to a particular application. The nAppID, nProgID, and nProcID are used for this purpose. The nInstID distinguishes multiple instances of the same program of the same application.

Data Type:EcTInt

Privilege:Private

Default Value:

**nMonState** - Monitoring State tells whether or not this application should be monitored by the subagent.

Data Type:MsTAgMonitorState

Privilege:Private

Default Value:

**nProcID** - The subagent keeps track of which processes belong to a particular program and which programs belong to a particular application. The nAppID, nProgID, and nProcID are used for this purpose. The nInstID distinguishes multiple instances of the same program of the same application.

Data Type:EcTInt

Privilege:Private

Default Value:

**nProgID** - The subagent keeps track of which processes belong to a particular program and which programs belong to a particular application. The nAppID, nProgID, and nProcID are used for this purpose. The nInstID distinguishes multiple instances of the same program of the same application.

Data Type:EcTInt

Privilege:Private

Default Value:

**pEventHdler** - event handler object. The event handler will have a pointer to the subagent's MsAgEventMgr client.

Data Type:MsAgEventHandler

Privilege:Private

Default Value:

**pMetHdler** - Metric handler objects.

Data Type:MsAgMetricHandler

Privilege:Private

Default Value:

**pRegistry** - MsAgRegistry. The EcAgManager can notify the Discoverer that it wants the subagent to start or stop polling for metrics and status.

Data Type:MsAgRegistryOD\_1\_0\*

Privilege:Private

Default Value:

**pfServer** - Pointer to the theServer of Process Framework type

Data Type:EcPfManagedServer

Privilege:Private

Default Value:

## Operations:

**AllocateRPCSpace** - This function allocates space from the rpc memory management scheme. This function should be called prior to objects being flattened to the MsTAgBuffer.

Arguments:EcTULongInt nSize, MsTAgBuffer pMsBuffer

Return Type:EcTVoid

Privilege:Private

**EcAgManager** - Constructor for EcAgManager. This construct creates the subagent's clients, MsAgEventMgr to send events to and MsAgDiscoverer to instruct the subagent to start monitoring this application. The constructor expects the executable name, application ID, program ID, process ID, instance ID, and a flag indicating to let the subagent know to start monitoring this process. The executable name is needed to derive the name of the application's configuration files. The processID and the instance ID will be determined by EcAgManager if nothing is specified. If the monitoring flag is not specified, the default is to not notify the subagent to start monitoring.

Arguments:RWCString &rsNewBinName, EcTInt nNewAppID, EcTInt nNewProgID, EcTInt nNewProcID=-1, EcTInt nNewInstID=-1, EcTBoolean nMonFlag=EcDFalse

Return Type:Void

Privilege:Public

**GetConfigMetric** - The subagent calls the following functions to retrieve a specific metric's information. The subagent must provide the index number of the metric vector, the level that the metric was registered, and an address to store the retrieved data into. This is an OODCE function and is only called by MSS.

Arguments:`/*[in]*/ EcTULongInt nMetricIndex, /*[in]*/ EcTAgMgmtLevel level, /*[out]*/ MsTAgBuffer metricBuff`

Return Type:EcTLongInt

Privilege:Public

PDL:EcTAgRetCode EcAgManager::GetConfigMetric (EcTUInt nMetricIndex, EcTAgMgmtLevel level, MsAgBuffer \*\*metricBuff)

{

```

// This function will default to an error condition if nLevel is not valid for this metric type.
// EcTAgRetCode retCode = EcEAgRcInvalidLevel;

// *vectPtr points to one of the RWOrdred vectors. This is used to reduce the number of
redundant
// calls.
// RWOrdered*vectPtr;

// if (level == EcEAgApplication)
{
// vectPtr = &appCfgMetVect;
// retCode = EcEAgRcSuccess;
}
// else if (level == EcEAgProgram )
{
// vectPtr = &prgCfgMetVect;
// retCode = EcEAgRcSuccess;
}

// if (retCode is successfully)
{
// Check if index is within bounds.
// if (nMetricIndex < vectPtr->entries ())
{
// Allocate space for the MsAgBuffer structure.
// *metricBuff = rpc_ss_allocate (sizeof (MsAgBuffer));

// Get the metric size to allocate the space necessary to store the actual data.
// (*metricBuff)->nSize =
((EcAgConfigMetric *) vectPtr[nMetricIndex]) -> binaryStoreSize () * 4;
// (*metricBuff)->pBuffer = rpc_ss_allocation ( (*metricBuff)->nSize);

// Convert the metric data into a machine independent stream of bytes and store it
// into nBuffer.
// ((EcAgConfigMetric *)vectPtr[nMetricIndex])->Flatten (*metricBuff)->nSize,
(*metricBuff)->nBuffer);
}
}
// return (retCode);
}

```

**GetConfigMetrics** - The subagent calls this function to get all the configuration metrics registered. This is an OODCE function and is only called by MSS.  
 Arguments:EcTAgMgmtLevel level, MsTAgBuffer metricBuff

```

Return Type:EcTLongInt
Privilege:Public
PDL:EcTAgRetCode EcAgManager::GetConfigMetrics (EcTAgMgmtLevel level,
MsAgBuffer **metricBuff)
{
// This function will default to an error condition if nLevel is not valid for this metric type.
// EcTAgRetCode retCode = EcEAgRcInvalidLevel;

// *vectPtr points to one of the RWOrdred vectors. This is used to reduce the number of
redundant
// calls.
// RWOrdered*vectPtr;

// if (level == EcEAgApplication)
{
// vectPtr = &appCfgMetVect;
// retCode = EcEAgRcSuccess;
}
// else if (level == EcEAgProgram )
{
// vectPtr = &prgCfgMetVect;
// retCode = EcEAgRcSuccess;
}

// if (retCode is successfully)
{
// Allocate space for the MsAgBuffer structure.
// *metricBuff = rpc_ss_allocate (sizeof (MsAgBuffer));

// Get the size of the entire vector.
// (*metricBuff)->nSize = vectPtr -> binaryStoreSize () * 4;
// (*metricBuff)->pBuffer = rpc_ss_allocation ( (*metricBuff)->nSize);

// Call our private member function to convert all the metric data into machine
// independent data stream and store it into metricBuff->pBuffer.
StoreFlattenedMetrics (vectPtr, metricBuff);
}
// return (retCode);
}

```

**GetFaultMetric** - The subagent calls the following functions to retrieve a specific metric's information. The subagent must provide the index number of the metric vector, the level that the metric was registered, and an address to store the retrieved data into. This is an OODCE function and is only called by MSS.

```

Arguments:[in]*/ EcTULongInt nMetricIndex, /*[in]*/ EcTAgMgmtLevel level, /
*[out]*/ MsTagBuffer metricBuff
Return Type:EcTLongInt
Privilege:Public
PDL:EcTAgRetCode    EcAgManager::GetFaultMetric    (EcTUInt    nMetricIndex,
EcTAgMgmtLevel level, MsAgBuffer **metricBuff)
{
// This function will default to an error condition if nLevel is not valid for this metric type.
// EcTAgRetCode retCode = EcEAgRcInvalidLevel;

// *vectPtr points to one of the RWOrdered vectors. This is used to reduce the number of
// redundant calls.
// RWOrdered*vectPtr;

// if (level == EcEAgProcess)
{
// vectPtr = &prcFaultMetVect;
// retCode = EcEAgRcSuccess;
}
// else if (level == EcEAgApplication)
{
// vectPtr = &appFaulMetVect;
// retCode = EcEAgRcSuccess;
}
// else if (level == EcEAgProgram )
{
// vectPtr = &prgFaultMetVect;
// retCode = EcEAgRcSuccess;
}

// if (retCode is successfully)
{
// Check if index is within bounds.
// if (nMetricIndex < vectPtr->entries ())
{
// Allocate space for the MsAgBuffer structure.
// *metricBuff = rpc_ss_allocate (sizeof (MsAgBuffer));

// Get the metric size to allocate the space neccessary to store the actual data.
// (*metricBuff)->nSize =
((EcAgFaultMetric *) vectPtr[nMetricIndex]) -> binaryStoreSize () * 4;
// (*metricBuff)->pBuffer = rpc_ss_allocation ( (*metricBuff)->nSize);

// Convert the metric data into a machine independent stream of bytes and store
// it into nBuffer.

```

```

// ((EcAgFaultMetric *)vectPtr[nMetricIndex])->Flatten (*metricBuff)->nSize ,
(*metricBuff)->nBuffer);
}
}
// return (retCode);
}

GetFaultMetrics - The subagent calls this function to get all the fault metrics registered. This is an OODCE function and is only called by MSS.
Arguments:EcTAgMgmtLevel level, MsTAgBuffer metricBuff
Return Type:EcTLongInt
Privilege:Public
PDL:EcTAgRetCode EcAgManager::GetFaultMetrics (EcTAgMgmtLevel level,
MsAgBuffer **metricBuff)
{
// This function will default to an error condition if nLevel is not valid for this metric type.
// EcTAgRetCode retCode = EcEAgRcInvalidLevel;

// *vectPtr points to one of the RWOrdred vectors. This is used to reduce the number of redundant
// calls.
// RWOrdered*vectPtr;

// if (level == EcEAgProcess)
{
// vectPtr = &prcFaultMetVect;
// retCode = EcEAgRcSuccess;
}
// else if (level == EcEAgApplication)
{
// vectPTr = &appFaulMetVect;
// retCode = EcEAgRcSuccess;
}
// else if (level == EcEAgProgram )
{
// vectPtr = &prgFaultMetVect;
// retCode = EcEAgRcSuccess;
}
// if (retCode is successfully)
{
// Allocate space for the MsAgBuffer structure.
// *metricBuff = rpc_ss_allocate (sizeof (MsAgBuffer));
}

```

```

// Get the size of the entire vector.
// (*metricBuff)->nSize = vectPtr -> binaryStoreSize () * 4;
// (*metricBuff)->pBuffer = rpc_ss_allocation ( (*metricBuff)->nSize);

// Call our private member function to convert all the metric data into machine
// independent data stream and store it into metricBuff->pBuffer.
StoreFlattenedMetrics (vectPtr, metricBuff);
}

// return (resultCode);
}

GetPerfMetric - The subagent calls this function to get a data of a particular performance
metric. This is a OODCE function and is only called by MSS.
Arguments:/*[in]*/ EcTULongInt nMetricIndex, /*[in]*/ EcTAvgMgmtLevel level, /
*[out]*/ MsTAvgBuffer metricBuff
Return Type:EcTLongInt
Privilege:Public
PDL:EcTAvgRetCode EcAgManager::GetPerfMetric (EcTUIInt nMetricIndex,
EcTAvgMgmtLevel nLevel, MsAgBuffer **metricBuff)
{
// This function will default to an error condition if nLevel is not valid for this metric type.
// EcTAvgRetCode resultCode = EcEAgRcInvalidLevel;

// *vectPtr points to one of the RWOrdred vectors. This is used to reduce the number of
redundant
// calls.
// RWOrdered*vectPtr;

// if (nLevel == EcAgProcess)
{
// vectPtr = &prcPerfMetVect;
// resultCode = EcEAgRcSuccess;
}
// else if (nLevel = EcEAgApplication)
{
// vectPtr = &appPerfMetVect;
// resultCode = EcEAgRcSuccess;
}

// if (resultCode is successfully)
{
// Check if index is within bounds.
// if (nMetricIndex < vectPtr->entries ())
{

```

```

// Allocate space for the MsAgBuffer structure.
// *metricBuff = rpc_ss_allocate (sizeof (MsAgBuffer));

// Get the metric size to allocate the space neccessary to store the actual data.
// (*metricBuff)->nSize =
((EcAgPerfMetric *) vectPtr[nMetricIndex]) -> binaryStoreSize () * 4;
// (*metricBuff)->pBuffer = rpc_ss_allocation ( (*metricBuff)->nSize);

// Convert the metric data into a machine independent stream of bytes and store it
// into nBuffer.
// vectPtr[nMetricIndex]->Flatten (*metricBuff)->nSize ,
(*metricBuff)->nBuffer);
}
}
// return (retCode);
}

```

**GetPerfMetrics** - The subagent calls the following functions to retrieve all metrics' information of a particular type. The subagent only needs to provide which level the metric was registered as and an address to store the retrieved data into. Notice that these functions are plural of those above. This is an OODCE function and is only called by MSS.

Arguments:EcTAgMgmtLevel level, MsTAgBuffer metricBuff

Return Type:EcTLongInt

Privilege:Public

PDL:EcTAgRetCode EcAgManager::GetPerfMetrics (EcTAgMgmtLevel nLevel,  
MsAgBuffer \*\*metricBuff)

{

// This function will default to an error condition if nLevel is not valid for this metric type.  
// EcTAgRetCode retCode = EcEAgRcInvalidLevel;

// \*vectPtr points to one of the RWOrred vectors. This is used to reduce the number of redundant

// calls.

// RWOrdered\*vectPtr;

// if (nLevel == EcAgProcess)

{

// vectPtr = &prcPerfMetVect;

// retCode = EcEAgRcSuccess;

}

// else if (nLevel = EcEAgApplication)

{

// vectPtr = &appPerfMetVect;

// retCode = EcEAgRcSuccess;

```

}

// if (retCode is successfully)
{
// Allocate space for the MsAgBuffer structure.
// *metricBuff = rpc_ss_allocate (sizeof (MsAgBuffer));

// Get the size of the entire vector.
// (*metricBuff)->nSize = vectPtr -> binaryStoreSize () * 4;
// (*metricBuff)->pBuffer = rpc_ss_allocation ( (*metricBuff)->nSize);

// Call our private member function to convert all the metric data into machine
// independent data stream and store it into metricBuff->pBuffer.
StoreFlattenedMetrics (vectPtr, metricBuff);
}
// return (retCode);
}

```

### **GetShutdownSec**

Arguments:`/*[in]*/EcTAgMgmtLevel nLevel`

Return Type:`EcTLongInt`

Privilege:Public

**Ping** - This function is used to reconnect to the subagent. The following function is distributed.

Arguments:

Return Type:`EcTLongInt`

Privilege:Public

**ProcessEvent** - This function takes the event passed in and depending on the category and severity, it sends the event to the subagent and depending on the category and loglevel, it logs the event to the MSS logger. This function is also used by the application to log events to its application log. No filtering is done here. The logType determine where the event is to be logged, either in MSS log file or the application log file.

Arguments:`EcAgEvent *pEvent, EcTAgLogType logType`

Return Type:`EcUtStatus`

Privilege:Public

PDL:`EcTVoid EcAgManager::ProcessEvent (EcAgEvent &event)`

{

// Create a temporary buffer to store EvAgEvent's data in a machine-independent

// fashion. Once this is done, call the EcAgEventManager's ProcessEvent fucntion.

// pEventMgr->ProcessEvent (MsAgBuffer \*eventBuff);

}

```

EcTVoid EcAgManager::Reconnect ()
{
    // Free any distributed objects created earlier.
    // if (pDiscover != NULL)
    {
        // delete [] pDiscoverer;
    }

    // if (pEventMgr != NULL)
    {
        // delete [] pEventMgr;
    }

    // Create new distributed objects.
    // pDiscoverer = new MsAgDiscovererOD_1_0;
    // pEventMgr = new MsAgEventMgrOD_1_0;

    // Send start monitoring.
    // nMonState = EcEAgMonitorON;
    // StartMonitoring ();
}

```

**Reconnect** - This is a distributed function called by the subagent to instruct this application to reconnect with the subagent.

Arguments:

Return Type:EcTLongInt

Privilege:Public

PDL:EcTVoid EcAgManager::Reconnect ()

```

{
    // Free any distributed objects created earlier.
    // if (pDiscover != NULL)
    {
        // delete [] pDiscoverer;
    }

    // if (pEventMgr != NULL)
    {
        // delete [] pEventMgr;
    }

    // Create new distributed objects.
    // pDiscoverer = new MsAgDiscovererOD_1_0;

```

```

// pEventMgr = new MsAgEventMgrOD_1_0;

// Send start monitoring.
// nMonState = EcEAgMonitorON;
// StartMonitoring ();
}

```

**RegisterMetric** - Adds the specified configuration metric in the appropriate configuration metric vector, based on the EcTAgMgmtLevel. Note that only applications and programs, can register a configuration metric. retCode is set to an error condition if level is not correct for a configuration metric.

Arguments:EcTAgMgmtLevel level, EcAgConfigMetric \*metric

Return Type:EcUtStatus

Privilege:Public

PDL:EcTAgRetCode EcAgManager::RegisterMetric (EcTAgMgmtLevel Level,  
EcAgConfigMetric \*metric)

{

// Adds the specified configuration metric in the appropriate configuration metric vector,  
based on

// the EcTAgMgmtLevel. Note that only applications and programs, can register

// a configuration metric. retCode is set to an error condition if level is not correct for a  
// configuration metric.

// EcTIntretCode = EcEAgInvalidLevel

```

// if (level == EcEAgApplication)
{
// appcCfgMetVect->insert (metric);
// retCode = EcEAgSuccess
}
// else if (level == EcEAgProgram )
{
// prgCfgMetVect->insert (metric);
// retCode = EcEAgSuccess
}
// return (retCode);
}

```

**RegisterMetric** - Adds the specified fault metric in the appropriate fault metric vector, based on the EcTAgMgmtLevel. Note that applications, programs, and processes can register a fault metric. retCode is set to an error condition if level is not correct for a fault metric.

Arguments:EcTAgMgmtLevel level, EcAgFaultMetric \*metric

Return Type:EcUtStatus

```

Privilege:Public
PDL:EcTAgRetCode    EcAgManager:::RegisterMetric    (EcTAgMgmtLevel    Level,
EcAgFaultMetric *metric)
{
// Adds the specified fault metric in the appropriate fault metric vector, based on
// the EcTAgMgmtLevel. Note that applications, programs, and processes can register
// a fault metric. retCode is set to an error condition if level is not correct for a fault metric.
// EcTIntretCode = EcEAgInvalidLevel

// if (level == EcEAgProcess)
{
// prcFaultMetVect->insert (metric);
// retCode = EcEAgSuccess
}
// else if (level == EcEAgApplication)
{
// appFaulMetVect->insert (metric);
// retCode = EcEAgSuccess
}
// else if (level == EcEAgProgram )
{
// prgFaultMetVect->insert (metric);
// retCode = EcEAgSuccess
}
// return (retCode);
}

```

**RegisterMetric** - Adds the specified performance metric in the appropriate performance metric vectPtr, based on the EcTAgMgmtLevel. Note that only applications and processes can register performance metrics. retCode is set to an error condition if level is not correct for a performance metric.

Arguments:EcTAgMgmtLevel level, EcAgPerfMetric \*metric

Return Type:EcUtStatus

Privilege:Public

```

PDL:EcTAgRetCode    EcAgManager:::RegisterMetric    (EcTAgMgmtLevel    level,
EcAgPerfMetric *metric)
{
// Adds the specified performance metric in the appropriate performance metric vector,
based on
// the EcTAgMgmtLevel. Note that only applications and processes can register
// performance
// metrics. retCode is set to an error condition if level is not correct for a performance
// metric.
// EcTIntretCode = EcEAgInvalidLevel;

```

```

// if (level == EcEAgProcess)
{
// prcPerfMetVect->insert (metric);
// retCode = EcEAgSuccess;
}
// else if (level == EcEAgApplication)
{
//appPerfMetVect->insert (metric);
// retCode = EcEAgSuccess
}

// return (retCode)
}

```

**Resume** - This function resumes a process at the given management level.

Arguments:/\*[in]\*/ EcTAgMgmtLevel level

Return Type:Void

Privilege:Public

PDL:EcTLongInt EcAgManager Resume ( EcTAgMgmtLevel nLevel,  
EcTLongInt gracefulFlag )

```

// {
//     MsTAgSuspendParam *param;
//     fill out the param struct
//     param = new MsTAgSuspendParam;
//     param->level = nLevel;
//     create a DCE thread, start the ResumeExec function and pass it param
//     delete param;
// }

```

**ResumeExec** - static thread function that start the resume

Arguments:pthread\_addr\_t

Return Type:Void

Privilege:Public

PDL:pthread\_addr\_t EcAgManager::ResumeExec (pthread\_addr\_t argument)

```

// {
//     MsTAgResumeParam *param = (MsTAgResumeParam *) argument;
//     fill in the param struct and call the process framework resume function
//     delete param
// }

```

### **SetCommonConfig**

Arguments:/\*[in]\*/MsTAgCommonCfg \*pCommCfg

Return Type:EcTLongInt

Privilege:Public

**SetConfigMetric** - The subagent uses this function to set a configuration metric's attributes. This is an OODCE function and is only called by MSS.

Arguments:  
/\*[in]\*/ EcTULongInt nMetricIndex, /\*[in]\*/ EcTAvgMgmtLevel nLevel, /  
\*[out]\*/ MsTAvgBuffer metricBuffer

Return Type:EcTLongInt

Privilege:Public

PDL:EcTAvgRetCode EcAgManager::SetConfigMetric (EcTInt nMetricIndex,  
EcTAvgMgmtLevel level, MsAgBuffer \*\*butPtr)

{

// This function will default to an error condition if nLevel is not valid for this metric type.  
// EcTAvgRetCode retCode = EcEAgRcInvalidLevel;

// \*vectPtr points to one of the RWOrdred vectors. This is used to reduce the number of redundant

// calls.

// RWOrdered\*vectPtr;

// if (nLevel == EcAgProgram)

{

// vectPtr = &prgCfgMetVect;

// retCode = EcEAgRcSuccess;

}

// else if (nLevel = EcEAgApplication)

{

// vectPtr = &appCfgMetVect;

// retCode = EcEAgRcSuccess;

}

// if (retCode is successful)

{

// Check if specified index is not out of bound.

// if (nMetricIndex < vector->entries ())

{

// Attempt to lock the file. If able to lock the file, then call the EcAgMetric's

// Restore function and compare the type of metric with the type that is in our

// Configuration vector. If the types are the same, then copy the new metric on

// top of the old metric. Unlock the metric when completed. Update retCode

// accordingly.

}

}

// return (retCode);

}

**SetPerfMetric** - The subagent uses this function to set a performance metric's attributes. This is an OODCE function and is only called by MSS.

Arguments:  
 /\*[in]\*/ EcTULongInt nMetricIndex, /\*[in]\*/ EcTAgMgmtLevel nLevel, /  
 \*[out]\*/ MsTAgBuffer metricBuffer

Return Type:EcTLongInt  
 Privilege:Public  
 PDL:EcTAgRetCode      EcAgManager::SetPerfMetric      (EcTInt      nMetricIndex,  
 EcTAgMgmtLevel level, MsAgBuffer \*\*butPtr)  
 {  
 // This function will default to an error condition if nLevel is not valid for this metric type.  
 // EcTAgRetCode retCode = EcEAgRcInvalidLevel;  
 // \*vectPtr points to one of the RWOrdred vectors. This is used to reduce the number of  
 redundant  
 // calls.  
 // RWOrdered\*vectPtr;  
 // if (nLevel == EcAgProcess)  
 {  
 // vectPtr = &prcPerfMetVect;  
 // retCode = EcEAgRcSuccess;  
 }  
 // else if (nLevel = EcEAgApplication)  
 {  
 // vectPtr = &appPerfMetVect;  
 // retCode = EcEAgRcSuccess;  
 }  
 // if (retCode != -1)  
 {  
 // Check if the specified index is not out of bound.  
 // if (nMetricIndex < prcPerfMetVect.entries ())  
 {  
 // Attempt to lock the file. If able to lock the file, then call the EcAgMetric's  
 // Restore function and compare the type of metric with the type that is in our  
 // Performance vector. If the types are the same, then copy the new metric on top  
 // of the old metric. Unlock the metric when completed. Update retCode  
 // accordingly.  
 }  
 }  
 // return (retCode);  
}

**SetupAgentMetrics** - This function retrieves information from this application's configuration file. These files are mandatory.

Arguments:RWCString, MsTAgCommonCfg&

Return Type:EcTVoid

Privilege:Private

**Shutdown** - This function invokes the appropriate shutdown method based on the EcTAgMgmtLevel passed in by the subagent. The subagent gives the number of seconds the application has to actually shutdown. The application needs to return from this call to notify the subagent of the time it really needs to shutdown. Since the subagent waits for the return number of seconds, the application developers may want to start a thread to shutdown. RetSec will be assigned by one of the Shutdown calls; therefore, no default is needed. This is an OODCE function and is only called by MSS.

Arguments:/\*[in]\*/ EcTAgMgmtLevel level, /\*[in]\*/ EcTLongInt nSeconds

Return Type:EcTLongInt

Privilege:Public

PDL:EcTInt EcAgManager::Shutdown (EcTAgMgmtLevel level, EcTInt nSeconds)

{

// This function invokes the appropriate shutdown method based on the EcTAgMgmtLevel passed

// in by the subagent. The subagent gives the number of seconds the application has to actually

// shutdown. The application needs to return from this call to notify the subagent of the time it

// really needs to shutdown. Since the subagent waits for the return number of seconds, the // application developers may want to start a thread to shutdown.

//

// retSec will be assigned by one of the Shutdown calls; therefore, no default is needed.

//

// EcTIntretSec;

// if (level == EcEAgMgmtProc)

{

// retSec = pShutdown->Process (nSeconds);

}

// else if (level == EcEAgMgmtApp)

{

// retSec = pShutdown->Application (nSeconds);

}

// else if (level == EcEAgMgmtProg)

{

// retSec = pShutdown->Program (nSeconds);

}

// else if (level == EcEgMgmtCOTS)

{

```

// retSec = pShutdown->COTS (nSeconds);
}

// If retSec is -1, then no shutdown function was provided; otherwise it's the number of
seconds
// this application needs to shutdown.
// return (retSec);
}

```

**ShutdownExec** - static thread function

Arguments:pthread\_addr\_t

Return Type:status pthread\_addr\_t

Privilege:Private

**StartMonitoring** - Call the Discoverer's Start function. This notifies the subagent to start monitoring this application. It also lets the subagent know that this application has started.

Arguments:

Return Type:EcUtStatus

Privilege:Public

PDL:EcTAgRetCode EcAgManager::StartMonitoring ()

{

// Call the Discoverer's Start function. This notifies the subagent of that this application is running

// and wants to be monitored.

// EcTAgRetCode retCode = EcEAgRcSuccess;

// DCEObjRefT myInfo;

// Only start the monitoring if it hasn't been done already.

// if (nMonState != EcTAgMonitorON)

{

// Fill myInfo with DCE specific data and pass the information over to the subagent's

// discoverer. If the call was successful, then set the nMonState to indicate that this

// application has been registered with the subagent and will be monitored.

// Update the retCode based on this call.

//if (pDiscoverer->Start (nAppID, nProgID, nProcId, nInstId, &myInfo) is successful )

{

// mMonState = EcTAgMonitorON;

}

}

// return (retCode);

}

**StopMonitoring** - Calls the Discoverer's Stop function. This notifies the subagent to stop monitoring this application and remove all information about this application from its

internal table.

Arguments:

Return Type:EcUtStatus

Privilege:Public

```
PDL:EcTAgRetCode EcAgManager::StopMonitoring ()
{
// Calls the Discoverer's Stop function. This notifies the subagent to stop monitoring this
// application and remove all information about this application from it's internal table.
// EcTAgRetCode retCode = EcEAgRcSuccess

// Only call Stop if this application is currently being monitored.
// if (nMonState == EcTAgMonitor)
{
// Call Discoverer's Stop function so that the subagent will clean all entries of this
// application. If this call is successful change nMonState to indicate that subagent has
// been notified. Update retCode based on this call.
// if (pDiscoverer->Stop (nAppID, nProgID, nProcID, nInstID) is successful)
{
// nMonState = EcTAgMonitorOff
}
}
// return (retCode);
}
```

**Suspend** - This function provides functionality to suspend a process. It invokes the appropriate suspend method based on the EcTAhMgmtLevel passed in by the subagent. The subagent gives the number of seconds the application has to suspend.

Arguments:`/*[in]*/ EcTAgMgmtLevel level, /*[in] */ EcTLongInt n Seconds`

Return Type:Void

Privilege:Public

```
PDL:EcTLongInt EcAgManage::Suspend(EcTAgMgmtLevel nLevel,
                                     EcTLongInt gracefulFlag)
```

```
// {
//   figure out the number of seconds required to suspend.
//   clean up
//   fill in the suspend param struct
//   MsTAgSuspendParam *param;
//   param = new MsTAgSuspendParam;
//   param->level = nLevel;
//   param->gracefulFlag = (EcTInt) gracefulFlag;
//
//   create a thread and start the suspendExec function.
// }
```

**SuspendExec** - static thread function that starts the suspend  
 Arguments:pthread\_addr\_t  
 Return Type:Void  
 Privilege:Public  
 PDL: pthread\_addr\_t EcAgManager::SuspendExec (pthread\_addr\_t argument)  
 // {  
 // MsTAgSuspendParam \*param = (MsTAgSuspendParam\*) argument;  
 // fill out the param struct  
 // pass it to the process framwork suspend function  
 // }

**~EcAgManager** - This method represents the destructor of the class.  
 Arguments:  
 Return Type:Void  
 Privilege:Public

### Associations:

The EcAgManager class has associations with the following classes:

- Class: EcUtLoggerRelA accesses
- Class: MsAgDeputyGate communicateswith
- Class: MsAgRegistry notifies
- Class: EcAgMetric registered/invokedby
- Class: MsAgMonitor sendsdatato
- Class: MsAgEventMgr sendseventsto
- Class: EcAgConfigFile usedby
- Class: EcAgException usedby
- Class: EcAgProxy usedby
- Class: EcAgShutdown usedby
- Class: MsAgMetricHandler usedby
- Class: ECSApplication uses/managedby
- Class: EcAgHostInfo uses

#### 4.1.3.14 EcAgMetric Class

Parent Class:Not Applicable

Public:No

Distributed Object:No

Purpose and Description:

This is an abstract class that defines the common operations such as (un-)Lock, save/restoreGuts. Its descendants contain performance/configuration/fault data.

## **Attributes:**

**owner** - This attribute specifies the owner of a metric. (agent or app)

**pLock** - This attribute is used to Lock/unLock a metric

**rsType** - This attribute contains the type.

## **Operations:**

**EcAgMetric** - This method represents the default constructor. It's only used by RWDECLARE and it's required by Rogue Wave.

Arguments:

**EcAgMetric** - This is the constructor that assigns the type.

Arguments:RWCString &rsNewType

**EcAgMetric**

Arguments:const EcAgMetric &metric

**Flatten**

Arguments:EcTInt nBufLen, EcTUChar \*szBuf

**GetOwner** - This method gets the ownership of a metric. (agent or app)

Arguments:

**GetType** - Gets rsType.

Arguments:

**Lock** - Lock pLock.

Arguments:

**Restore**

Arguments:EcTInt nBufLen, EcTUChar \*szBuf

**SetOwner**

Arguments:MsTAgMetricOwner newOwner

**SetType**

Arguments:RWCString& rsNewType

**TryLock** - Try to lock pLock.

Arguments:

**UnLock** - Unlock pLock.

Arguments:

**binaryStoreSize** - Return the number of bytes used by the virtual function saveGuts (RWFile&) to store an object.

Arguments:

**compareTo**

Arguments:const RWCollectable \*c

**isEqual**

Arguments:const RWCollectable \*c

**operator=**

Arguments:const EcAgMetric &metric

**restoreGuts**

Arguments:RWFile& f

**restoreGuts**

Arguments:RWvistream& stream

**saveGuts**

Arguments:RWFile& f

**saveGuts**

Arguments:RWvostream& stream

**~EcAgMetric** - This is the default destructor.

Arguments:

### Associations:

The EcAgMetric class has associations with the following classes:

Class: ECSApplication creates

Class: EcAgManager registered/invokedby

Class: MsAgDeputyGate uses

#### 4.1.3.15 EcAgNamedList Class

Parent Class:Not Applicable

Public:No

Distributed Object:No

Purpose and Description:

this class represents a RW singely linked list that has a text name

### **Attributes:**

**list** - linked list of values

**rsName** - name of linked list

### **Operations:**

**EcAgNamedList** - default constructor

Arguments:

**EcAgNamedList** - constructor accepting a default name and linked list

Arguments:RWCString rsNewName, RWSlistCollectables \*pList

**GetList** - get the linked list of values

Arguments:EcUtStatus &status

**GetName** - get the name of the list

Arguments:EcUtStatus &status

**SetList** - set the linked list of values

Arguments:RWSListCollectables \*pList

**SetName** - set the name of the list

Arguments:RWCString rsNewName

**isEqual** - RW reimplemented method

Arguments:const RWCollectables \*c

**-EcAgNamedList** - default destructor

Arguments:

### **Associations:**

The EcAgNamedList class has associations with the following classes:

Class: EcAgConfigFile usedby

#### **4.1.3.16 EcAgPatternVec Class**

Parent Class:Not Applicable

### **Attributes:**

None

#### **Operations:**

None

#### **Associations:**

The EcAgPatternVec class has associations with the following classes:

None

#### **4.1.3.17 EcAgPerfMetric Class**

Parent Class:EcAgMetric

Public:No

Distributed Object:No

Purpose and Description:

This metric contains performance data.

#### **Attributes:**

**nFaultThreshold** - fault threshold

**nMaxThreshold** - Maximum threshold

**nMinThreshold** - Minimum Threshold.

**nRearmFaultThreshold** - Rarm fault threshold

**nRearmMaxThreshold** - Rarm maximum threshold.

**nRearmMinThreshold** - Rarm minimum threshold.

**nValue** - Performance metric value.

#### **Operations:**

**EcAgPerfMetric** - This method represents the default constructor. It's only used by RWDECLARE and it's required by Rogue Wave.

Arguments:

**EcAgPerfMetric** - This method represents the constructor which assigns the parameters passed in to its class attributes.

Arguments:RWCString &rsNewType, EcTInt nnewValue, EcTInt nNewFaultThreshold,  
EcTInt nNewMaxThreshold, EcTInt nNewMinThreshold, EcTInt  
nNewRearmFaultThreshold, EcTInt nNewRearmMaxThreshold, EcTInt  
nNewRearmMinThreshold

**EcAgPerfMetric** - This method represents the copy constructor.

Arguments:const EcAgPerfMetric &metric

**GetFaultThreshold** - Get nFaultThreshold.

Arguments:

**GetMaxThreshold** - Get maximum threshold.

Arguments:

**GetMinThreshold** - Get minimum threshold.

Arguments:

**GetRearmFaultThreshold** - Gets rearm fault threshold.

Arguments:

**GetRearmMaxThreshold** - Gets rearm maximum threshold.

Arguments:

**GetRearmMinThreshold** - Gets rearm minimum threshold.

Arguments:

**GetValue** - Get nValue.

Arguments:

**SetFaultThreshold**

Arguments:EcTInt nNewFaultThreshold

**SetMaxThreshold**

Arguments:EcTInt nNewMaxThreshold

**SetMinThreshold**

Arguments:EcTInt nNewMinThreshold

**SetRearmFaultThreshold**

Arguments:EcTInt nNewRearmFaultThreshold

**SetRearmMaxThreshold**

Arguments:EcTInt nNewRearmMaxThreshold

**SetRearmMinThreshold**

Arguments:EcTInt nNewRearmMinThreshold

**SetValue**

Arguments:EcTInt nnewValue

**binaryStoreSize** - Return the number of bytes used by the virtual function saveGuts (RWFile&) to store an object.

Arguments:

**operator <<**

Arguments:ostream& stream, EcAgPerfMetric& metric

**operator >>**

Arguments:istream& stream, EcAgPerfMetric& metric

**operator=**

Arguments:const EcAgPerfMetric& metric

**restoreGuts** - This method represents the restoreGuts operation for a file.

Arguments:RWFile& f

**restoreGuts**

Arguments:RWvistream& stream

**saveGuts** - This method represent saveGuts for a file.

Arguments:RWFile& f

**saveGuts**

Arguments:RWvostream& stream

**~EcAgPerfMetric** - Default destructor.

Arguments:

**Associations:**

The EcAgPerfMetric class has associations with the following classes:

Class: MsAgPerfEvent contains

#### 4.1.3.18 EcAgProxy Class

Parent Class:Not Applicable

Public:Yes

Distributed Object:Yes

Purpose and Description:

This object class is primarily for COTS' manageability. It includes the MSS

instrumentation class library to enable the manageability of the COTS product. The front-end of this object is the MSS instrumentation code. The back-end of it is the interface to the COTS. It is unique to every COTS. In security management, the logs of COTS are monitored by this object. If an security event occurs, this object has to detect the incident and send out an event notification to the MsAgSubagent.

#### **Attributes:**

**Lock** - lock for the proxyEntry ordered

Data Type:DCEPthreadMutex

Privilege:Private

Default Value:

**entryVector** - RW Ordered of proxy entries

Data Type:RWOrdered

Privilege:Private

Default Value:

**pCOTSMgrFactory** - A pointer to the COTSMgrFactory object.

Data Type:EcAgCOTSMgrFactory\*

Privilege:Private

Default Value:

**rsName** - Name of the proxy agent.

Data Type:RWCString

Privilege:Private

Default Value:

#### **Operations:**

**CreateCDSName** - Creates a CDS the object uses to register into the CDS.

Arguments:

Return Type:RWCString

Privilege:Private

**EcAgProxy**

Arguments:RWCString rsNewName

Return Type:Void

Privilege:Public

**Listen** - Listens for incoming requests to be monitored.

Arguments:

Return Type:EcUtStatus

Privilege:Public

**LoadPreviousManagers** - connects to COTSmanagers that the agent was connected to before it shutdown (or possibly died).

Arguments:

Return Type:EcTVoid

Privilege:Private

**Ping** - listens for incomming requests to be monitored

Arguments:

Return Type:EcTVoid

Privilege:Public

**ResumeManager** - This distributed method is executed by ProxyComm. It resumes the manager specified by the ProcID passed in.

Arguments:EcTChar\* szProxyName, EcTLongInt nNewAppID, EcTLongInt nNewProgID, EcTLongInt nNewProcID, EcTLongInt nNewInstID

Return Type:EcTVoid

Privilege:Public

PDL:EcTVoid ECAgProxyManager::ResumeManager( EcTChar \* szProxyName,  
EcTLongInt nAppID, EcTLongInt nProgID,  
EcTLongInt nProcID, EcTLongInt nInstID

```
// {  
//     create a temporary entry to find the actual object in the suspended vector  
//     lock the entry vector and enter the entry to resumed in the manager vector  
//     tell the manager to resume that entry  
//     unlock  
// }
```

**SaveEntryVector** - save the entry vector to disk

Arguments:

Return Type:EcTVoid

Privilege:Private

**SaveSuspendedEntryVector** - This function saves the suspended entry to a file so that it can be read back in at resume time.

Arguments:

Return Type:EcTVoid

Privilege:Private

PDL:EcTVoid ECAgProxy::SaveSuspendedEntryVector()

```
//{
//     copy the entry to be suspended from the entry vector
//     into the suspended entry vector
// }
```

**SetManagerFactory** - Sets the default manager factory object.

Arguments:EcAgCOTSMgrFactory \*pNewCOTSMgrFactory

Return Type:EcTVoid

Privilege:Public

**SpawnManagerProcess** - executes a single manager passed in a MsAgProxyParam which is in turn, passed in as a EcTVoid\*. This method is executed by a newly spawned thread

Arguments:pthread\_addr\_t pData

Return Type:statuc pthread\_addr\_t

Privilege:Private

**StartManager** - rpc method to request a COTS process to be monitored

Arguments:EcTUChar\* szProxyName, EcTLongInt nNewAppID, EcTLongInt nNewProgID, EcTLongInt nNewProcID, EcTLongInt nNewInstID

Return Type:EcTVoid

Privilege:Private

**StopManager** - rpc method to request a COTS process to be monitored

Arguments:EcTUChar\* szProxyName, EcTLongInt nNewAppID, EcTLongInt nNewProgID, EcTLongInt nNewProcID, EcTLongInt nNewInstID

Return Type:EcTVoid

Privilege:Public

**SuspendManager** - This distributed method is executed by ProxyComm. It suspends the manager specified by the ProcID passed in.

Arguments:EcTUChar\* szProxyName, EcTLongInt nNewAppID, EcTLongInt nNewProgID, EcTLongInt nNewProcID, EcTLongInt nNewInstID

Return Type:EcTVoid

Privilege:Public

PDL:EcTVoid EcAgProxy::SuspendManager (EcTUChar \* szProxyName,

EcTLongInt nAppID, EcTLongInt nProgID,

EcTLongInt nProcID, EcTLongInt nInstID

// {

// create a temporary entry to find the actual object in the manager vector

// lock the entry vector and find the entry to suspend

// once the entry has been found tell the manager to suspend that entry

// unlock

// }

## Associations:

The EcAgProxy class has associations with the following classes:

- Class: COTS manages
- Class: EcAgCotsLog monitors
- Class: EcAgCOTSManager usedby
- Class: EcAgCOTSMgrFactory usedby
- Class: EcAgManager usedby
- Class: MsAgSubAgent uses

#### **4.1.3.19 EcAgShutdown Class**

Parent Class:Not Applicable

Public:No

Distributed Object:No

Purpose and Description:

The EcAgShutdown should be used by the ECS application to handle graceful shutdowns for their application triggered by the subagent. There are four virtual functions to allow ECS application to shutdown as an application, a program, or a process. It will be up to the application to close any shared resources and shutdown the appropriate processes when shutting down the application or program. The fourth shutdown function is to shutdown the COTS. When the EcAgManager executes one of the calls below, it passes the number of seconds the application has to shutdown the appropriate process(es). If the application needs more time than a specified one, it returns the number of seconds it needs to shutdown to the subagent. Since the subagent will be waiting for the return value of the shutdown command, when these functions are redefined, it should create a thread to actually shutdown the application so that the function can return the number of seconds it needs to shutdown back to the subagent. Currently the virtual functions return -1 so the subagent will know that no shutdown procedure was redefined.

#### **Attributes:**

None

#### **Operations:**

##### **Application**

Arguments:EcTLongInt nTimeInterval

##### **COTS**

Arguments:EcTLongInt nTimeInterval

##### **EcAgShutdown - Default Constructor.**

Arguments:

##### **Process**

Arguments:EcTLongInt nTimeInterval

### **Program**

Arguments:EcTLongInt nTimeInterval

**~EcAgShutdown** - Default destructor.

Arguments:

### **Associations:**

The EcAgShutdown class has associations with the following classes:

Class: ECSApplication instantiates

Class: EcAgManager usedby

Class: MsAgDeputyGate usedby

### **4.1.3.20 EcAgTuple Class**

Parent Class:Not Applicable

Public:No

Distributed Object:No

Purpose and Description:

This class is used by EcAgEvent to represent a tuple of type and value.

### **Attributes:**

**rsType** - Tuple type.

**rsValue** - Tuple value.

### **Operations:**

**EcAgTuple** - This is the default constructor. Required by RogueWave because of RWDECLARE\_COLLECTABLE. It's only used by RWDECLARE.

Arguments:

#### **EcAgTuple**

Arguments:RWCString &rsNewType, RWCString &rsnewValue

**EcAgTuple** - This method represents the copy constructor.

Arguments:const EcAgTuple &tuple

**GetType** - Get rsType.

Arguments:

**GetValue** - Gets rsValue.

Arguments:

**SetType**

Arguments:RWCString &rsNewType

**SetValue**

Arguments:RWCString &rsnewValue

**binaryStoreSize** - This method represents the binary store size.

Arguments:

**operator <<** - This method streams an object to ostream.

Arguments:ostream&, EcAgTuple&

**operator =**

Arguments:EcAgTuple& tuple

**operator >>** - This method streams an object from istream.

Arguments:istream&, EcAgTuple&

**restoreGuts** - This method represents the restoreGuts operation for a file.

Arguments:RWFile& f

**restoreGuts** - This method represents the restoreGuts operation for a stream.

Arguments:RWvistream& stream

**saveGuts** - This method represents the saveGuts function for a file.

Arguments:RWFile& f

**saveGuts** - This method represents the saveGuts operation for a stream.

Arguments:RWvostream& stream

**-EcAgTuple** - Default destructor.

Arguments:

## Associations:

The EcAgTuple class has associations with the following classes:

EcAgEvent (Aggregation)

#### **4.1.3.21 EcUtLoggerRelA Class**

Parent Class:Not Applicable

Public:No

Distributed Object:No

Purpose and Description:

This class is a public class that provides a logging capability used by the Management Agent Services logging function.

#### **Attributes:**

None

#### **Operations:**

None

#### **Associations:**

The EcUtLoggerRelA class has associations with the following classes:

Class: MsAgEventMgr accessedby

Class: EcAgManager accesses

Class: EcAgHostInfo uses

#### **4.1.3.22 MsAgAgent Class**

Parent Class:Not Applicable

Public:No

Distributed Object:No

Purpose and Description:

This managed object class is the master (SNMP) agent on the host. It listens to port 161 to receive SNMP requests from management applications. It also sends SNMP traps to management applications when certain events occur. MSS requires this master agent be extensible to support subagents. The agent performs authentication and authorization validations on incoming requests. If the requested MIB variables are in MIB II, it performs the functions requested. If the MIB variables are not in MIB II but in registered MIB extensions, it passes the request to the subagent which supports that particular MIB extension.

#### **Attributes:**

None

**Operations:**

None

**Associations:**

The MsAgAgent class has associations with the following classes:

Class: MsAgEncps communicateswith

Class: MsAgSubAgent communicateswith

**4.1.3.23 MsAgAppMIB Class**

Parent Class:Not Applicable

Public:No

Distributed Object:No

Purpose and Description:

The SNMP MIB extension defined for applications, especially for ECS applications being managed. Three groups of objects are defined. Due to the number of variables, they are not all enumerated in this document. Each one has general attributes defined based on the functional areas of management applications such as configuration, performance, fault, and security, wherever applicable. The application-specific attributes can be defined by application developers in an extensible way using tables. Attributes: Application - the application to manage. Program - the program in the application package being managed. Process - the process of a program in the application being managed.

**Attributes:**

None

**Operations:**

None

Associations:

The MsAgAppMIB class has associations with the following classes:

Class: MsAgSubAgent uses

**4.1.3.24 MsAgCfgFileInfo Class**

Parent Class:Not Applicable

Public:No

Distributed Object:No

Purpose and Description:

This class is used to put into a temporary vector so that when MsAgDiscoverer compares what's installed and what's in the subagent's table, it can do this easily.

## **Attributes:**

**nID** - ID of the application or program

**nParentID** - ID of the application's or the program's parent

**rsFullName** - Full path and filename of the configuration file

**rsMajorVersion** - Major version

**rsMinorVersion** - Minor version

**rsMode** - Mode

## **Operations:**

**GetFileName** - Gets the full path and filename

Arguments:

**GetId** - Returns the ID of the application/program

Arguments:

**GetMajorVersion** - Returns the major version

Arguments:

**GetMinorVersion** - Returns the minor version

Arguments:

**GetMode** - Returns the mode

Arguments:

**GetParentId** - Returns the parent ID of the application/program

Arguments:

**MsAgCfgFileInfo** - Default constructor

Arguments:

**MsAgCfgFileInfo** - Constructor that reads from config file and fills info.

Arguments: RWCString rsNewFileName, EcTAgMgmtLevel level, RWCString rsNewMode

**MsAgCfgFileInfo** - Temp constructor for searching

Arguments: EcTInt nNewID, RWCString rsNewMjVer, RWCString rsNewMnVer, RWCString rsNewMode, EcTInt nNewParentID=-1

**isEqual** - RW function for searching  
Arguments:Const RWCollectable \*tmp

**~MsAgCfgFileInfo** - Default destructor  
Arguments:

#### **Associations:**

The MsAgCfgFileInfo class has associations with the following classes:

Class: MsAgDiscoverer uses

#### **4.1.3.25 MsAgDeputy Class**

Parent Class:Not Applicable

Public:No

Distributed Object:No

Purpose and Description:

This object is used both by the management applications and by the subagent. The management applications can send Set requests to the subagent through this object. The subagent can send event notifications to this object so an SNMP trap can be emitted to management framework.

#### **Attributes:**

**QueueLock** - lock for the queue

**cmCommunity** - community

**cmPeerName** - peer name

**eventQueue** - queue of events that have not been sent as traps

**mutex** - dce thread mutex needed by condition variable

**pConditionVar** - condition variable

**pSession** - pointer to the current SNMP session

#### **Operations:**

**Execute** - start the deputy

Arguments:

**MsAgDeputy** - This is the constructor for this class.

Arguments:

**ProcessEvent** - process a single event (distributed method)

Arguments: MsTAgBuffer \*pHostBuffer, MsTAgEventDelimiter eventType, MsTAgBuffer \*pEventBuff

**~MsAgDeputy** - This is the destructor for this class.

Arguments:

### Associations:

The MsAgDeputy class has associations with the following classes:

Class: MsAgDeputyGate communicateswith

Class: MsAgEventMgr communicateswith

Class: MsAgEventEntry usedby

Class: MsAgSnmpPdu usedby

#### 4.1.3.26 MsAgDeputyGate Class

Parent Class: Not Applicable

Public: No

Distributed Object: Yes

Purpose and Description:

This class communicates with the Deputy class using OODCE for secure and reliable communication. This class receives set-requests from the Deputy.

### Attributes:

**pBndVct** - pointer to the binding vector

Data Type: MsAgMgmtVector

Privilege: Private

Default Value:

**pTblMgr** - This attribute represents a pointer to a table manager.

Data Type: MsAgTblMgr\*

Privilege: Private

Default Value:

**systemLock** - pointer to the system lock (locked before any calls to the system command)

Data Type: DCEPthreadMutex

Privilege: Private

Default Value:

## **Operations:**

**GetChildIDList** - returns a linked list of child ID's.

Arguments:EcTInt nID, EcTInt nInstanceID, EcTAgMgmtLevel nParentMgmtLevel, EcTAgMgmtLevel nChildMgmtLevel

Return Type:RWSlistCollectables\*

Privilege:Protected

**GetExecInfo** - returns the ID, instanceID, and management level of the specified app, program, or process table

Arguments:MsTAgSNMPTblID nTableID, EcTInt nTableIndex, EcTInt &nID, EcTInt &nInstanceID, EcTAgMgmtLevel &nMgmtLevel

Return Type:EcTInt

Privilege:Protected

**GetShutdownSeconds** - returns the number of seconds the application, program, or process will need to shutdown gracefully

Arguments:EcTLongInt nTableID, EcTLongInt nTableIndex

Return Type:EcTLongInt

Privilege:Public

**MsAgDeputyGate** - default constructor

Arguments:MsAgTblMgr \*pNewTblMgr, MsAgMgmtBindingVector \*pNewBndVct

Return Type:Void

Privilege:Public

**PerformAction** - depending on the value of nAction, performs either a shutdown or a get seconds to shutdown. 0 = Shutdown, 1 = GetShutdownSecs

Arguments:EcTInt nID, EcTInt nInstanceID, EcTAgMgmtLevel nMgmtLevel, EcTAgMgmtLevel nStopMgmtLevel, EcTInt nShutdownType, EcTInt nAction

Return Type:EcTInt

Privilege:Protected

**ResumeExecutable** - Resumes the specified application, program, or process.

Arguments:EcTLongInt nTableID, EcTLongInt nTableIndex, EcTLongInt nInstanceID, MsTAgStringPtr szCommandLine

Return Type:EcTVoid

Privilege:Public

PDL:EcTLongInt EcAgProxy::ResumeExecutable ( EcTLongInt nTableID,  
EcTLongInt nTableIndex, EcTLongInt nInstID )

// {

// access the table based on the management level (MsEAgStAppTbl

// or MsEAgStProgTbl)

// lock table

```
//      get the entry from the table  
//      recreate the system call command  
//      unlock the table  
//      systemLock.Lock();  
//      restart the executable  
//      systemLock.Unlock();  
// }
```

**SetInt** - sets the value of an integer object within an SNMP table

Arguments:EcTLongInt nTableID, EcTLongInt nTableIndex, EcTLongInt nAttributeIndex, EcTInt nValue

Return Type:EcTVoid

Privilege:Public

**SetString** - sets the value of a string object within an SNMP table

Arguments:EcTLongInt nTableID, EcTLongInt nTableIndex, EcTLongInt nAttributeIndex, MsTAgStringPtr szValue

Return Type:EcTVoid

Privilege:Public

**SetValue** - sets the value of either a string or an integer attribute within any of the SNMP tables

Arguments:EcTLongInt nTableID, EcTLongInt nTableIndex, EcTLongInt nAttributeIndex, MsTAgSNMPEntry \*pData

Return Type:EcTVoid

Privilege:Protected

**StartExecutable** - starts an application, program, or process. If InstanceID is equal to -1, then a unique instance id is generated

Arguments:EcTLongInt nTableID, EcTLongInt nTableIndex, EcTLongInt nInstanceID, MsTAgStringPtr szCommandLine

Return Type:EcTVoid

Privilege:Public

**StopExecutable** - executes the shutdown method of the specified application, program, or process.

Arguments:EcTLongInt nTableID, EcTLongInt nTableIndex nShutdownType

Return Type:EcTLongInt

Privilege:Public

**SuspendExecutable** - Suspends the specified application, program, or process.

Arguments:EcTLongInt nTableID, EcTLongInt nTableIndex nShutdownType

Return Type:EcTLongInt

Privilege:Public

PDL:EcTLongInt EcAgProxy::SuspendExecutable (EcTLongInt nTableID,

```

EcTLongInt nTableIndex, EcTLongInt nSuspendSec)
// {
//   get ID and instance of executable
//   suspend all affected processes
//   find the table entry using the SNMPTbl access tale function
//   lock the table
//   set time to suspend
//   unlock
// }

```

### **Associations:**

The MsAgDeputyGate class has associations with the following classes:

- Class: EcAgManager communicateswith
- Class: MsAgDeputy communicateswith
- Class: EcAgShutdown usedby
- Class: MsAgEventHandler usedby
- Class: EcAgMetric uses
- MsAgSubAgent (Aggregation)

#### **4.1.3.27 MsAgDiscoverer Class**

Parent Class:Not Applicable

Public:No

Distributed Object:Yes

Purpose and Description:

ECS applications contact the discoverer through the EcAgManager when they start up so they can be monitored. When the SubAgent first comes up it tries to discover all applications, programs and processes that are already running.

### **Attributes:**

**pTblMgr** - This attribute represents a pointer to a a table manager.

Data Type:MsAgTblMgr\*

Privilege:Private

Default Value:

### **Operations:**

**Activate** - This function adds mode to the active mode file and issues a DiscoverNow operation to discover applications in the recently added mode.

Arguments:RWCString rsNewMode

Return Type:EcUtStatus

Privilege:Public

```

PDL:EcUtStatus MsAgDiscoverer::Activate (RWCString rsNewMode )
//{
//    EcUtStatus status;
//    open the active mode file
//    check if the new mode already exists
//    if not, add the new mode to the active mode file
//    call DiscoverNow to discover recently added mode
//    DiscoverNow();
//    return status
//}

```

**AddFile** - Adds files to the vector

Arguments:RWCString &file, RWOrdered &vector, EcTAgMgmtLevel level, const EcTChar \*pMode

Return Type:EcUtStatus

Privilege:Private

**AddToTable** - Add application, program to the table

Arguments:EcTAgMgmtLevel level, RWOrdered &vector, MsAgSNMPTbl \*pTable

Return Type:EcTVoid

Privilege:Private

**AdjustEntry** - Adjusts the index when an entry is removed from a table

Arguments:Ec TInt newEntries, Ec TInt &realEntries, Ec TInt &index

Return Type:EcTVoid

Privilege:Private

**CompareTable** - Compares the vector and the application, program tables

Arguments:EcTAgMgmtLevel level, RWOrdered &vector, MsAgSNMPTbl \*pTable, Ec TInt notifyAllFlag=FALSE

Return Type:EcTVoid

Privilege:Private

**Deactivate** - This function removes the mode from the active mode file and issues a DiscoverNow operation to remove the applications from the deactivated mode.

Arguments:RWCString rsNewMode

Return Type:EcUtStatus

Privilege:Public

PDL: EcUtStatus MsAgDiscoverer::Deactvate (RWCString rsMode )

```

// {
//    EcUtStatus status;
//    open the active mode file
//    look for the mode name rsMode
//    if it doesn't exist : ERROR
//    else remove it from the active mode file
}

```

```
//      call DiscoverNow to undiscover removed mode  
//      DiscoverNow();  
//      return status  
// }
```

**DiscoverNow** - Periodic discovery of new applications and programs

Arguments:

Return Type:EcTVoid

Privilege:Public

**FillTableEntry** - Reads from the config file and fills the table entry

Arguments:EcTAgMgmtLevel level, MsAgTeVerExec \*pEntry, MsAfCfgFileInfo \*pFile

Return Type: \*pFile

Privilege:Private

**FindInstalled** - Finds all installed programs and applications and puts them into the corresponding vectors

Arguments:RWOrdered &appVector, RWOrdered &prgVector

Return Type:EcTVoid

Privilege:Private

**MsAgDiscoverer** - default constructor assigns values.

Arguments:MsAgTblMgr \*pNewTblMgr

Return Type:Void

Privilege:Public

**Rediscover** - Notifies HPOV of every installed application and program

Arguments:

Return Type:EcTVoid

Privilege:Public

**ReportEvent** - Send cfg file error event

Arguments:EcTAgEventCategory nNewCategory, EcTInt nNewType,

EcTAgEventSeverity nNewSeverity, RWCString &fileName, const EcTChar \*pMode

Return Type:EcUtStatus

Privilege:Private

**ReportEvent** - This function sends a topology change event.

Arguments:EcTAgEventCategory nNewCategory, EcTInt nNewType,

EcTAgEventSeverity nNewSeverity, EcTAgMgmtLevel level, MsAgTeVerExec \*pEntry

Return Type:EcUtStatus

Privilege:Public

**ReportEvent** - This function sends a version mismatch event.

Arguments:EcTAgEventCategory nNewCategory, EcTInt nNewType,

EcTAgeEventSeverity nNewSeverity, EcTAgeMgmtLevel level, MsAgCfgFileInfo  
&tblFile, MsAgCfgFileInfo &vectFile  
Return Type:EcUtStatus  
Privilege:Public

**~MsAgDiscoverer** - This method represents the destructor.

Arguments:

Return Type:Void  
Privilege:Public

### **Associations:**

The MsAgDiscoverer class has associations with the following classes:

Class: MsAgCfgFileInfo uses  
MsAgSubAgent (Aggregation)

### **4.1.3.28 MsAgEncps Class**

Parent Class:Not Applicable

Public:No

Distributed Object:No

Purpose and Description:

The encapsulator enables the MsAgAgent to communicate with non-Peer agents.

### **Attributes:**

None

### **Operations:**

None

### **Associations:**

The MsAgEncps class has associations with the following classes:

Class: MsAgAgent communicateswith

### **4.1.3.29 MsAgEventEntry Class**

Parent Class:Not Applicable

Public:No

Distributed Object:No

Purpose and Description:

This class is used by the Deputy to queue events received by the subagent.

#### **Attributes:**

**nEventType** - describes the type of event

**pEvent** - pointer to the event

**pHostInfo** - pointer to the host info

#### **Operations:**

**GetEvent** - returns a pointer to the event

Arguments:

**GetHostInfo** - returns a pointer to the host info

Arguments:

**MsAgEventEntry** - default constructor

Arguments:

**MsAgEventEntry** - constructor accepting all class attributes

Arguments:EcAgEvent \*pNewEvent, EcAgHostInfo \*pNewHostInfo,  
MsTAgeventDelimiter nNewEventType

**~MsAgEventEntry** - destructor

Arguments:

#### **Associations:**

The MsAgEventEntry class has associations with the following classes:

Class: MsAgDeputy usedby

#### **4.1.3.30 MsAgEventHandler Class**

Parent Class:Not Applicable

Public:No

Distributed Object:No

Purpose and Description:

This class handles event for EcAgManager. It filters events based on severity level and it also logs events to the application and MSS log files.

#### **Attributes:**

**mssCfg** - Structure with common buffers

**pAppLog** - Logger object for application log

**pAppLogEnabled** - Configuration metrics read in by MsAgMetricHandler

**pAppMaxLogSize** - Configuration metrics read in by MsAgMetricHandler

**pEventMgr** - Client to the subagent's MsAgEventMgr

**pMssEventFilter** - Configuration metrics read in by MsAgMetricHandler

**pMssLog** - Logger object for MSS log

### **Operations:**

**MsAgEventHandler** - Constructor for MsAgEventHandler

Arguments:rsCDSName, MsAgMetricHandler \*pMetHandler, MsTAgCommonCfg \*pNewMssCfg, RWCString rsMode, RWCString rsSite

**ProcessAppEvent** - This function processes application

Arguments:EcAgEvent \*pEvent

**ProcessMssEvent** - This function processes mss events

Arguments:EcAgEvent \*pEvent

**SetConfiguration** - This function sets the subagent's configuration metrics

Arguments:MsTAgCommonCfg \*pCfg

**~MsAgEventHandler** - Default destructor

Arguments:

### **Associations:**

The MsAgEventHandler class has associations with the following classes:

Class: MsAgDeputyGate usedby

#### **4.1.3.31 MsAgEventMgr Class**

Parent Class:Not Applicable

Public:No

Distributed Object:Yes

Purpose and Description:

The Event Manager logs and processes events. To log, it needs to access the Logger from

CSS. Events will be processed by means of the Deputy for secure delivery to HP-OpenView.

### Attributes:

**pDeputy** - Deputy is the intermediate step between the EventManager and HP-OpenView. OODCE will be used to send events to the Deputy who will pass the event to HP-OpenView.

**pHostBuff** - This attribute contains a pointer to the host information that's stored in an MsTAgBuffer.

**pHostInfo** - This attribute represents a pointer to the host information object.

**pLogger** - This attribute represents a pointer to a logger. Here is the CSS logger which will store events in a logfile.

### Operations:

#### **LogEvent**

Arguments:EcAgEvent \*pEvent, MsTAgEventDelimiter eventType

#### **MsAgEventMgr**

Arguments:uuid\_t \*obj

**ProcessEvent** - This function is called by the EcAgManager to send events to the Deputy. This function processes an event for the EcAgManager MsAgMonitor. The MsTAgBuffer structure will be passed the MsAgDeputy on the HP-OpenView side.

Arguments:/\* [in] \*/ MsTAgBuffer \*pBuffer

PDL:EcTVoid MsAgEventMgr::ProcessEvent (MsAgBuffer \*pBuffer)

{

//EcAgEvent event;

// Convert pBuffer to an Event object using the 'restore' function (opposite of flatten function)

// Store the Event in event;

//call ManageEvent with the event type and the event.

// ManageEvent (EcEAgEvent, event);

}

EcTVoid MsAgEventMgr::LogEvent (MsTAgEventCode eventType, EcAgEvent event)

{

// Stream eventType and event into the CSS logger;

// pLogger << eventType << event;

```
}
```

**ProcessEvent** - This function is called by the subagent and is not a distributed function. The events are logged if indicated and sent to the Deputy if indicated. This function processes an event for the MsAgMonitor. The EventMgr class receives a MsAgPerfEvent object. This object will first be logged in the CSS logger. Then it will be converted into a MsTAgBuffer and sent to HP-OpenView by calling the other ProcessEvent function provided by MsAgEventMgr.

Arguments:MsAgPerfEvent\*

```
PDL:EcTVoid MsAgEventMgr:ProcessEvent (MsTAgEventCode eventType, EcAgEvent &event)
```

```
{
```

```
//Create two variable which will be used by the flatten function to hand a
```

```
//MsAgBuffer buffHostInfo;
```

```
//MsAgBuffer buffEvent;
```

```
// Log event.
```

```
// LogEvent (eventType, event);
```

```
// Call EcAgHostInfo and EcAgEvent's flatten functions to store their data in a machine-independent
```

```
// fashion and store the results in buffHostInfo and buffEvent. A reason for sending structure instead of
```

```
//objects is because OODCE can only process structures.
```

```
// pDeputy->ProcessEvent(buffHostInfo, eventType, buffEvent);
```

```
}
```

```
// Log event.
```

```
// LogEvent (eventType, event);
```

```
// Call EcAgHostInfo and EcAgEvent's flatten functions to store their data in a machine-independent
```

```
// fashion and store the results in buffHostInfo and buffEvent. A reason for sending structure instead of
```

```
//objects is because OODCE can only process structures.
```

```
// pDeputy->ProcessEvent(buffHostInfo, eventType, buffEvent);
```

```
}
```

## **ProcessEvent**

Arguments:EcAgEvent \*pEvent, MsTAgEventDelimiter eventType

**~MsAgEventMgr** - This method represents the default destructor.

**Arguments:**

**Associations:**

The MsAgEventMgr class has associations with the following classes:

Class: EcUtLoggerRelA accessedby  
Class: MsAgDeputy communicateswith  
Class: EcAgManager sendseventsto  
Class: MsAgMonitor talksto  
Class: EcAgHostInfo usedby  
Class: MsAgSubAgentConfig uses  
MsAgSubAgent (Aggregation)

#### **4.1.3.32 MsAgIntConfigMetric Class**

Parent Class:EcAgConfigMetric

Public:No

Distributed Object:No

Purpose and Description:

This class is a specialized class of EcAgConfigMetric. Since EcAgConfigMetric's type is RWCString, This class converts RWCString to EcTInt so that getting the value will first check if it can be converted. This saves time and effort to get value as integer. The GetValueAsInt was overloaded to return nValue. This class also contains a specialed class of MsAgIntConfigMetric, called MsAgUpdateCfgMetric. This metric is tied directly with the update interval seconds for MsAgProcSShotInfo.

**Attributes:**

**nValue** - Integer value of the configuration metric

**Operations:**

**ConvertToInt** - Converts RWCString to integer

Arguments:RWCString rsStrValue, EcUtStatus &status

**GetValueAsInt** - Get value as integer

Arguments:

**MsAgIntConfigMetric** - Constructor for MsAgIntConfigMetric

Arguments:RWCString &rsNewType, RWCString &rsNewValue

**MsAgIntConfigMetric** - Copy constructor for EcAgConfigMetric

Arguments:const EcAgConfigMetric &metric

**SetValue** - Set value as integer

Arguments:EcTInt nnewValue

**SetValue** - Set value as RWCString

Arguments:RWCString &rsnewValue

**~MsAgIntConfigMetric** - default destructor

Arguments:

### **Associations:**

The MsAgIntConfigMetric class has associations with the following classes:

None

### **4.1.3.33 MsAgMetVector Class**

Parent Class:Not Applicable

Public:No

Distributed Object:No

Purpose and Description:

This class is used to store EcAgMetrics in an RWOrdered array. This class also provides functions to Flatten and Restore the entire RWOrdered array into a machine-independent stream of data.

### **Attributes:**

None

### **Operations:**

**Flatten** - Converts this MsAgMetVector object into a machine-independent stream of data.

INPUTS nBufLen - Length of buffer to store converted data. \*pBuf - Address to store converted data.

Arguments:EcTInt nBufLen, EcTUChar \*pBuf

**MsAgMetVector** - Default constructor.

Arguments:

**Restore** - Converts the machine-independent data back into the MsAgMetVector object.

INPUTS nBufLen - length of the buffer containing data. \*pBuf - address of where the converted data is.

Arguments:EcTInt nBufLen, EcTUChar \*pBuf

**~MsAgMetVector** - Default destructor.

**Arguments:**

**Associations:**

The MsAgMetVector class has associations with the following classes:

EcAgManager (Aggregation)

#### **4.1.3.34 MsAgMetricHandler Class**

Parent Class:Not Applicable

Public:No

Distributed Object:No

Purpose and Description:

This class handles all the reading and creating of metrics for EcAgManager. It also contains the vectors that stores the different types of metrics in their applicable management level. The application specific metrics are registered to these vectors using EcAgManager's wrapper function.

**Attributes:**

**appCfgFile** - This applications's configuration filename and path

**appCfgMetVect** - Application metric vector

**appFaultMetVect** - Application metric vector

**appName** - This application's name

**appPerfMetVect** - Application metric vector

**pProcInfo** - Process' snap-shot information

**prcFaultMetVect** - Process metric vector

**prcPerfMetVect** - Process metric vector

**prgCfgMetVect** - Program metric vector

**prgFaultMetVect** - Program metric vector

**progCfgFile** - This program's configuration filename and path

**Operations:**

**FindConfigMetric** - Returns an address to a configuration metric in one of our vectors  
Arguments:EcTAgMgmtLevel level, RWCString rsMetType, EcUtStatus &status

**GetAllConfigMetrics** - Gets all the config metrics from the specified vector  
Arguments:EcTAgMgmtLevel level, MsAgMetVector &vector

**GetAllFaultMetrics** - Gets all the fault metrics from the specified vector  
Arguments:EcTAgMgmtLevel level, MsAgMetVector &vector

**GetAllPerfMetrics** - Gets all the performance metrics from the specified vector  
Arguments:EcTAgMgmtLevel level, MsAgMetVector &vector

**GetAndCheckPerfMetric** - Gets and registers performance metrics  
Arguments:EcAgConfigFile &cfgFile, RWCString rsSearchStr, MsTAgProcPMTypemType

**GetAppName** - Inline function to get application name  
Arguments:

**GetConfigVector** - Gets the appropriate configuration vector  
Arguments:EcTAgMgmtLevel level, EcUtStatus &status

**GetFaultVector** - Gets the appropriate fault vector  
Arguments:EcTAgMgmtLevel level, EcUtStatus &status

**GetMetric** - Puts the specified performance metrics into the specified address  
Arguments:EcTAgMgmtLevel level, EcAgPerfMetric &metric, EcTInt nMetricIndex

**GetMetric** - Puts the specified config metrics into the specified address  
Arguments:EcTAgMgmtLevel level, EcAgFaultMetric &metric, EcTInt nMetricIndex

**GetMetric** - Puts the specified fault metrics into the specified address  
Arguments:EcTAgMgmtLevel level, EcAgConfigMetric &metric, EcTInt nMetricIndex

**GetPerfVector** - Gets the appropriate performance vector  
Arguments:EcTAgMgmtLevel level, EcUtStatus &status

**MsAgMetricHandler** - Constructor for MsAgMetricHandler  
Arguments:RWCString rsProgName, RWCString rsSite, RWCString rsMode

**ReadAppCfg** - Reads application configuration file and creates metrics to put into our vectors.  
Arguments:RWCString rsCfgFile, RWCString rsSite, RWCString rsMode

**ReadProgCfg** - Read program configuration file and creates metrics to put into our

vectors

Arguments:RWCString rsCfgFile, RWCString rsSite, RWCString rsMode

**RegisterMetric** - Inserts the specified performance metric to the correct vector  
Arguments:EcTAgMgmtLevel level, EcAgPerfMetric \*pMetric

**RegisterMetric** - Inserts the specified configuration metric to the correct vector  
Arguments:EcTAgMgmtLevel level, EcAGConfigMetric \*pMetric

**RegisterMetric** - Inserts the specified fault metric to the correct vector  
Arguments:EcTAgMgmtLevel level, EcAgFaultMetric \*pMetric

**RegisterProcMetrics** - Registers process-specific metrics  
Arguments:EcAgConfigFile &cfgFile

**RemoveAppCfgMetrics** - Removes application configuration metrics that we added  
Arguments:

**RemovePrcPerfMetrics** - Removes process performance metrics that we added  
Arguments:

**SaveMetric** - Saves current performance metric to file  
Arguments:EcTAgMgmtLevel level, EcAgPerfMetric \*pMetric

**SaveMetric** - Saves current configuration metric to file  
Arguments:EcTAgMgmtLevel level, EcAGConfigMetric \*pMetric

**SetMetric** - Sets the specified performance metric into the specified address  
Arguments:EcTAgMgmtLevel level, EcAgPerfMetric &metric, EcTInt nMetricIndex

**SetMetric** - Sets the specified config metrics into the specified address  
Arguments:EcTAgMgmtLevel level, EcAGConfigMetric &metric, EcTInt nMetricIndex

**~MsAgMetricHandler** - Destructor for MsAgMetricHandler  
Arguments:

## Associations:

The MsAgMetricHandler class has associations with the following classes:

Class: EcAgManager usedby

Class: MsAgPathFinder uses

#### **4.1.3.35 MsAgMgmtBindingHandle Class**

Parent Class:Not Applicable

Public:No

Distributed Object:No

Purpose and Description:

This is a (single) binding handle. (subagent side)

#### **Attributes:**

**pObjRef**

#### **Operations:**

**AddConfigMetrics**

Arguments:MsAgMgmtHandle \*pMgrHdle, EcTAgMgmtLevel level

**AddFaultMetrics**

Arguments:MsAgMgmtHandle \*pMgrHdle, EcTAgMgmtLevel level

**FindMetric**

Arguments:EcAgConfigMetric \*pMetric, EcTInt nId, EcTint nParentId, EcTint nInstId, MsAgSNMPTbl \*pTable

**FindMetric**

Arguments:EcAgFaultMetric \*pMetric, EcTInt nId, EcTInt nParentId, EcTInt nInstID, MsAgSNMPTbl \*pTable

**FindMetric**

Arguments:EcAgPerfMetric \*pMetric, EcTInt nId, EcTInt nParentId, EcTInt nInstId, MsAgSNMPTbl \*pTable

**GetObjRecT** - Gets the DCEObjRef structure passed in through the constructor.

Arguments:

**MsAgMgmtBindingHandle** - Constructor with DCEObjRefT defined. Makes a private copy of pNewObjRef.

Arguments:DCEObjRefT \*pNewObjRef

**MsAgMgmtBindingHandleThis** - This is the default constructor.

Arguments:

**binaryStoreSize** - This function determines the size of the MsAgMgmBindingHandle object.

Arguments:

**restoreGuts** - This function streams data from the file into the MsAgMgmtBindingHandle's attributes.

Arguments:RWFile& rfFile

**restoreGuts** - This function takes data from an output stream and stores the data into the MsAgMgmtBindingHandle's attributes.

Arguments:RWvistream& riStream

**saveGuts** - This function streams MsAgMgmtBindingHandle's attribute to the file specified.

Arguments:RWFile& rfFile

**saveGuts** - This function streams the MsAgMgmtBindingHandle attributes to an output stream.

Arguments:RWvostream roStream

**~MsAgMgmtBindingHandle** - This is the default destructor.

Arguments:

### Associations:

The MsAgMgmtBindingHandle class has associations with the following classes:

MsAgMgmtBindingVector (Aggregation)

#### 4.1.3.36 MsAgMgmtBindingVector Class

Parent Class:Not Applicable

Public:No

Distributed Object:No

Purpose and Description:

This is a RogueWave array with Locking and unLocking functionalities.

### Attributes:

**pLock** - Mutex to lock and unlock this binding vector

**rsBindFile** - Binding file name

### Operations:

**AddConfigMetrics**

Arguments:MsAgMgmtHandle \*pMgrHdle, EcTAgMgmtLevel level

**AddDynamicEntry**

Arguments:EcTAgMgmtLevel level, MsAgMgmtHandle \*pMgrHdle

**AddEntrytToTables**

Arguments:MsAgMgmtHandle \*pMgrHdle

**AddFaultMetrics**

Arguments:MsAgMgmtHandle \*pMgrHdle, EcTAgMgmtLevel level

**AddMgmtHandle**

Arguments:MsAgMgmtHandle \*pHdle

**AddPerfMetrics**

Arguments:MsAgMgmtHandle \*pMgrHdle, EcTAgMgmtLevel level

**AddProcEntry**

Arguments:MsAgMgmtHandle \*pMgrHdle

**FindMetric**

Arguments:EcAgPerfMetric \*pMetric, EcTInt nId, EcTInt nParentId, EcTInt nInstId, MsAgSNMPTbl \*pTable

**FindMetric**

Arguments:EcAgFaultMetric \*pMetric, EcTInt nId, EcTInt nParentId, EcTInt nInstID, MsAgSNMPTbl \*pTable

**Lock** - Locks pThreadMutex before using the binding vector.

Arguments:

**MsAgMgmtBindingVector** - This constructs the MsAgMgmtBindingVector object.

Arguments:

**MsAgMgmtBindingVector** - Constructor. Pass in the binding file path and name

Arguments:RWCString &bfName

**RemoveMgmtHandle**

Arguments:MsAgMgmtHandle \*pHdle

**Unlock** - Unlocks pThreadMutex after using the binding vector.

Arguments:

**clearAndDestroy** - Removes the binding file prior to destroying the vector's contents.

Arguments:

**insert** - Overloaded RW functions that inserts/removes handle to/from vector and updates the binding file. This can function may through an exception.

Arguments:RWCollectable\* pHdle

**remove** - Overloaded RW functions that inserts/removes handle to/from vector and updates the binding file. This can function may through an exception.

Arguments:const RWCollectable\* pHdle

**~MsAgMgmtBindingVector** - This destroys the MsAgMgmtBindingVector object.

Arguments:

### **Associations:**

The MsAgMgmtBindingVector class has associations with the following classes:

Class: MsAgRegistry uses

#### **4.1.3.37 MsAgMgmtHandle Class**

Parent Class:MsAgMgmtBindingHandle

Public:No

Distributed Object:No

Purpose and Description:

This class is a specialized MsAgMgmtBindingHandle that connects with an EcAgManager. This object will be saved in the MsAgMgmtBindingVector.

### **Attributes:**

**nAppID** - application ID

**nInstID** - instance ID

**nMaxRetries** - Maximum times to try to bind with the EcAgManager. Maximum number of retries to connect with EcAgManager server

**nProcID** - process ID

**nProgID**

**pMgmtObject** - Pointer to the EcAgManager client

**realFlag** - Flag to indicate if the real constructor was used

**rsMode** - mode

**rtPIDSec** - time of process

**Operations:**

**Bind**

Arguments:

**GetAppID** - returns application ID

Arguments:

**GetInstID** - returns instance ID

Arguments:

**GetMgmtHandle** - returns management handle

Arguments:

**GetMode** - returns mode

Arguments:

**GetProcID** - returns proc id

Arguments:

**GetProcStartTime** - returns process start time

Arguments:

**GetProgID**

Arguments:

**MsAgMgmtHandle**

Arguments:

**MsAgMgmtHandle** - constructor which assigns values

Arguments:EcTInt nNewAppID, EcTInt nNewProgID, EcTInt nNewProcID, EcTInt nNewInstID

**MsAgMgmtHandle** - Constructor with the object reference and PIDs

Arguments:DCEObjRefT \*pNewObjRef, EcTInt nNewAppID, EcTInt nNewProgID, EcTInt nNewProcID, EcTInt nNewInstID, RWCString rsNewMode, EcTInt nNewRwTime, EcTInt nNewMaxRetries=10

**binaryStoreSize** - overloaded RW function.

**Arguments:**

**isEqual** - overloaded RW function.  
Arguments:const RWCollectable\* c

**restoreGuts** - overloaded RW function.  
Arguments:RWFile &f

**restoreGuts** - overloaded RW function.  
Arguments:RWvistream& stream

**saveGuts** - overloaded RW function.  
Arguments:RWFile &f

**saveGuts** - overloaded RW function.  
Arguments:RWvostream& stream

**~MsAgMgmtHandle**  
Arguments:

### **Associations:**

The MsAgMgmtHandle class has associations with the following classes:  
None

#### **4.1.3.38 MsAgMonitor Class**

Parent Class:Not Applicable

Public:Yes

Distributed Object:Yes

Purpose and Description:

MsAgMonitor is spawned by the MsAgSubagent. It provides the local polling capability to monitor resources being managed. This can avoid the costly remote polling done by management applications while still being able to monitor the state of the resources. If error conditions occurred, it informs the agent to send a trap to the management application(s). The scope of this local polling is the host. The time interval applies to all the resources that are monitored by this object. This class also provides the capability for monitoring transient processes. Applications may specify that transient processes that they create be monitored for their presence until the monitoring is no longer required. Start and stop indications are needed from applications. Between the start and stop requests, the transient process is monitored. In the event that a monitored process fails, a notification is sent to the interested application.

### **Attributes:**

**pBindingVector** - pointer to binding vector

**pTblMgr** - This attribute represents a pointer to a table manager.

### **Operations:**

**CheckProcessState** - checks process state

Arguments:EcTInt pid

**CheckStatus** - checks status

Arguments:EcTInt index

**CheckThresholds** - checks thresholds

Arguments:MsAgTePerf \*pEntry, MsAgMgmtHandle \*pMgrHdle

**MsAgMonitor**

Arguments:MsAgTblMgr \*pTblMgr, MsAgMgmtBindingHandle \*pMgmtBindingHandle

**PollPerformance** - polls performance

Arguments:

**PollPerformanceTable** - poll performance table

Arguments:MsAgSNMPTbl \*pTable

**PollStatus** - This method checks if all processes registered are still running.

Arguments:

**~MsAgMonitor** - This method represents the constructor of the object.

Arguments:

### **Associations:**

The MsAgMonitor class has associations with the following classes:

Class: MsAgPerfEvent createdby

Class: MsAgPortMonitor receivesrequestsfrom

Class: EcAgManager sendsdatato

Class: MsAgEventMgr talksto

### **4.1.3.39 MsAgPathFinder Class**

Parent Class:Not Applicable

Public:No

Distributed Object:No

Purpose and Description:

This classn builds the path and filename of subagent related files and CDS names

**Attributes:**

**modeFlag** - Flag to indicate whether they have specified mode

**rsMode** - Mode name

**rsSite** - Site name

**siteFlag** - Flag to indicate whether they have specified site

**Operations:**

**GetAgentCDSName** - Gets the agent's CDS name

Arguments:EcUtStatus &status

**GetAgentDataFile** - Gets the agent data file

Arguments:EcUtStatus &status, RWCString dataFileName

**GetAgentLog** - Gets the MMS log path and filename

Arguments:EcUtStatus &status

**GetAppCfgFile** - Gets the application cfg path and filename

Arguments:EcUtStatus &status, RWCString prgName

**GetAppLog** - Gets the application's log path and filename

Arguments:EcUtStatus &status, RWCString appName

**GetCfgDir** - Gets the configuration directory

Arguments:EcUtStatus &status

**GetCfgModeDir** - Gets the configuration's subdirectories with mode

Arguments:EcUtStatus &status, const EcTChar \*pNewMode=NULL

**GetProgCfgFile** - Gets the program cfg path and filename

Arguments:EcUtStatus &status, RWCString prgName

**MsAgPathFinder** - Constructor for names with Site and Mode

Arguments:RWCString rsNewSite, RWCString rsNewMode

**MsAgPathFinger** - Constructor for names without Site or Mode

Arguments:

**SetMode** - Sets mode

Arguments:RWCString newMode

**SetSite** - Sets site

Arguments:RWCString newSite

**~MsAgPathFinder** - Destructor

Arguments:

### **Associations:**

The MsAgPathFinder class has associations with the following classes:

Class: MsAgMetricHandler uses

#### **4.1.3.40 MsAgPerfEvent Class**

Parent Class:EcAgEvent

Public:No

Distributed Object:No

Purpose and Description:

This class represents an event containing performance data. Only the subagent can create performance events.

### **Attributes:**

**nNumPerfMetrics** - Number of performance metrics

**perfMetrics**

### **Operations:**

**GetNumPerfMetrics** - Returns the number of performance metrics in the linked list.

There's no set function because 'insert perf metric' will increment nNumPerfMetrics.

Arguments:

**GetPerfMetric** - Gets the first performance metric from the linked list.

Arguments:

**GetPerfMetrics** - Returns the linked list of performance metrics.

Arguments:

**InsertPerfMetric**

Arguments:EcAgPerfMetric &newPerfMetric

**MsAgPerfEvent** - default constructor

Arguments:

**MsAgPerfEvent** - copy constructor

Arguments:const MsAgPerfEvent &event

**MsAgPerfEvent**

Arguments:Cat,Typ,Sev,Csci, opt:MgNm,Mg,STyp,Ssys,App,Prog,Inst,Proc,Trans, TransP

**binaryStoreSize** - Return the number of bytes used by the virtual function saveGuts (RWFile&) to store an object.

Arguments:

**operator <<**

Arguments:ostream& stream, MsAgPerfEvent& event

**operator >>**

Arguments:istream& stream, MsAgPerfEvent& event

**operator=**

Arguments:const MsAgPerfEvent& event

**restoreGuts**

Arguments:RWFile& f

**restoreGuts**

Arguments:RWvistream& stream

**saveGuts**

Arguments:RWFile& f

**saveGuts**

Arguments:RWvostream& stream

**~MsAgPerfEvent** - This is the default destructor.

Arguments:

### Associations:

The MsAgPerfEvent class has associations with the following classes:

Class: EcAgPerfMetric contains

Class: MsAgMonitor createdby

#### **4.1.3.41 MsAgPortMonitor Class**

Parent Class:Not Applicable

Public:No

Distributed Object:No

Purpose and Description:

/The PortMonitor listens to SMUX requests from the Master Agent, which is a COTS called Peer.

#### **Attributes:**

None

#### **Operations:**

##### **Execute**

Arguments:pthread\_addr\_t pThread

**MsAgPortMonitor** - This method represents the constructor.

Arguments:EcTInt NewSecs, EcTInt nNewMicrosecs

**~MsAgPortMonitor** - This method represents the destructor.

Arguments:

#### **Associations:**

The MsAgPortMonitor class has associations with the following classes:

Class: MsAgTblMgr accessedby

Class: MsAgMonitor receivesrequestsfrom

MsAgSubAgent (Aggregation)

#### **4.1.3.42 MsAgProcInfo Class**

Parent Class:Not Applicable

Public:No

Distributed Object:No

Purpose and Description:

This class provides a mechanism to retrieve a information from the same snap-shot for the dependent performance metric classes of the EcAgManager. This class is used to ensure that dependent metrics have values from the same snap-shot.

#### **Attributes:**

**nCpu** - cpu usage

**nDiskIO** - disk io

**nID** - process id

**nInRPCCalls** - in rpc calls

**nInRPCPkts** - in rpc packets

**nMem** - memory usage

**nNumOfThreads** - number of threads

**nOutRPCCalls** - out rpc calls

**nOutRPCPkts** - out rpc packets

**nPageSize /\* HP \*/** - page size

**rtTimeOfProcess** - time of process

## Operations:

**GetCPU**

Arguments:EcUtStatus &status

**GetDiskIO** - gets disk io

Arguments:EcUtStatus &status

**GetMem**

Arguments:EcUtStatus &status

**GetNumOfThreads**

Arguments:EcUtStatus &status

**GetRPCInCalls**

Arguments:EcUtStatus &status

**GetRPCInPkts**

Arguments:EcUtStatus &status

**GetRPCOutCalls**

Arguments:EcUtStatus &status

**GetRPCOutPkts**

Arguments:EcUtStatus &status

**MsAgProcInfo**

Arguments:EcTInt nNewID=-1

**OpenProcFIlle**

Arguments:

**~MsAgProcInfo** - Default destructor.

Arguments:

**Associations:**

The MsAgProcInfo class has associations with the following classes:

EcAgManager (Aggregation)

**4.1.3.43 MsAgProcPerfMetric Class**

Parent Class:EcAgPerfMetric

Public:No

Distributed Object:No

Purpose and Description:

This class is a special performance metric that contains either cpu, memory, various RPC information, the number of threads, or the number of disk I/O.

**Attributes:**

**pInfo** - This attribute points to the snapshot data.

**pmType** - type of performance metric

**Operations:**

**GetValue** - returns the value

Arguments:

**MsAgProcPerfMetric** - default constructor

Arguments:MsTAgProcPMTypE, MsAgProcSShotInfo \*pNewInfo, RWCString &rsNewInfo, RWCString &rsNewType, EcTInt nNEewValue, EcTInt nNewFaultThreshold, EcTInt nNewMaxThreshold, EcTInt nNewMinThreshold, EcTInt nNewRearmFaultThreshold, EcTInt nNewRearmMAXThreshold, EcTInt nNewRearmMinThreshold

**~MsAgProcPerfMetric** - default destructor

Arguments:

### **Associations:**

The MsAgProcPerfMetric class has associations with the following classes:

Class: MsAgProcSShotInfo uses  
EcAgManager (Aggregation)

### **4.1.3.44 MsAgProcSShotInfo Class**

Parent Class:MsAgProcInfo

Public:No

Distributed Object:No

Purpose and Description:

This class provides a mechanism to retrieve information from the same snap-shot for the classes that inherited from the performance metric class of the EcAgManager.

### **Attributes:**

**nUpdateInterval** - update interval

**pLock** - dce thread mutex

**rtLastUpdateTime** - last update time

### **Operations:**

**GetCPU** - returns the cpu usage

Arguments:EcUtStatus &status

**GetDiskIO** - returns the disk io

Arguments:EcUtStatus &status

**GetInRPCCalls** - returns the number of in RPC calls

Arguments:EcUtStatus &status

**GetInRPCPkts** - returns number of in rpc packets

Arguments:EcUtStatus &status

**GetMem** - returns the memory usage

Arguments:EcUtStatus &status

**GetNumOfThreads** - returns the number of threads

Arguments:EcUtStatus &status

**GetOutRPCCalls** - returns the number of out rpc calls

Arguments:EcUtStatus &status

**GetOutRPCPkts** - returns the number of out rpc packets

Arguments:EcUtStatus &status

**GetUpdateInterval** - gets update interval

Arguments:

**GetUpdateTime** - returns update time

Arguments:

**Lock** - locks class

Arguments:

**MsAgProcSShotInfo** - default constructor

Arguments:EcTInt nNewUpdateInterval, EcTInt nNewID=-1

**SetProcSShotInfo** - sets the process snatshot info

Arguments:

**SetUpdateInterval** - returns update interval

Arguments:EcTInt nNewUpdateInterval, EcUtStatus &status

**SetUpdateTime** - sets update time

Arguments:RWTime rtNewTime

**UnLock** - unlocks class

Arguments:

**UpdateValues** - updates all values

Arguments:

**~MsAgProcSShotInfo** - default destructor

Arguments:

## Associations:

The MsAgProcSShotInfo class has associations with the following classes:

Class: MsAgProcPerfMetric uses

#### **4.1.3.45 MsAgRegistry Class**

Parent Class:Not Applicable

Public:No

Distributed Object:Yes

Purpose and Description:

This class is used by the subagent to discover applications and programs installed on the host. It finds this by examining a well-known directory where application and program configuration files exist. It reads the vital information from those files and enters them into the subagent's internal tables. This class is also used by the EcAgManager to notify the subagent to start and stop monitoring the managed object which contains the EcAgManager.

#### **Attributes:**

**pBindVector** - Pointer to the binding vector

Data Type:MsAgMgmtBindingVector\*

Privilege:Private

Default Value:

**pEventMgr** - Pointer to the Event Manager

Data Type:MsAgEventMgr\*

Privilege:Private

Default Value:

**pTblMgr**

Data Type:MsAgTblMgr\*

Privilege:Private

Default Value:

#### **Operations:**

**AddDynamicEntry** - Private functions that retrieve information from the EcAgManager and fills in the internal subagent's tables.

Arguments:EcTAgMgmtLevel level, MsAgMgmtHandle \*pMgrHdle

Return Type:EcUtStatus

Privilege:Private

**AddEntryToTables** - Private functions that retrieve information from the EcAgManager and fills in the internal subagent's tables.

Arguments:MsAgMgmtHandle \*pMgrHdle

Return Type:EcUtStatus

Privilege:Private

**AddMgmtHandle** - Private function to update the bind vector file

Arguments:MsAgMgmtHandle \*pHdle

Return Type:EcUtStatus

Privilege:Private

**AddPerfMetrics** - Private functions that retrieve information from the EcAgManager and fills in the internal subagent's tables.

Arguments:MsAgMgmtHandle \*pMgrHdle, EcTAgMgmtLevel level

Return Type:EcUtStatus

Privilege:Private

**AddProcEntry** - Private functions that retrieve information from the EcAgManager and fills in the internal subagent's tables.

Arguments:MsAgMgmtHandle \*pMgrHdle

Return Type:EcUtStatus

Privilege:Private

### **MsAgRegistry**

Arguments:MsAgTblMgr \*pNewTblMgr, MsAgMgmtBindingVector

\*pNewMgmtBindVector, uuid\_t\* obj

Return Type:Void

Privilege:Public

**ReBind** - Private function to rebind to the managed process

Arguments:MsAgMgmtHandle \*pMgr

Return Type:EcUtStatus

Privilege:Private

**ReEstablishConnection** - Private function reestablish connection with every managed prc

Arguments:

Return Type:EcTVoid

Privilege:Private

**RemoveMgmtHandle** - Private function to update the bind vector file

Arguments:MsAgMgmtHandle \*pHdle

Return Type:EcUtStatus

Privilege:Private

**ReportEvent** - Private function for handling errors

Arguments:MsAgMgmtHandle \*pMgr, EcTAgEventCategory nNewCategory, EcTInt nNewType, EcTAgEventSeverity nNewSeverity, EcTAgMgmtLevel level = EcEAGMgmtProc

Return Type:EcUtStatus

Privilege:Private

**RestoreMgmtHandle** - This function is used when a process is being resumed. It deletes handle from the suspended vector and inserts it back into the binding vector.

Arguments:MsAgMgmtHandle \*pHdle

Return Type:EcUtStatus

Privilege:Private

PDL:EcUtStatus RestoreMgmtHandle( MsAgMgmtHandle \*pHdle )

```
// {
```

```
// EcUtStatus status;
```

```
// remove the entry from the suspended vector and add it to the binding vector.
```

```
// pSuspendedVector->remove(pHdle);
```

```
// pBindVector->insert(pHdle);
```

```
// return (status);
```

```
// }
```

## Resume

Arguments:EcTInt nNewAppID, EcTInt nNewProg, EcTInt nNewProcID, EcTInt nNewInstID, MsAgStringBuff sNewMode, EcTULongInt nNewRWTime, DCEObjRefT \*pNewDCbjRef

Return Type:EcTVoid

Privilege:Public

PDL:EcTVoid MsAgRegistry::Resume(EcTULongInt nAppID, EcTULongInt nProgID,  
EcTULongInt nProcID, EcTULongInt nInstID, EcTULongInt nSubSystem,  
EcTULongInt sMode, EcTULongInt nRWTime, DCEObjRefT \*pObjRef )

```
//{
```

```
// Create a resume parameter and copy this discoverer object and create a copy of the  
// ObjRef.
```

```
//
```

```
// Create a thread to add the resumed process to subagent's internal tables.
```

```
//
```

```
//}
```

**ResumeThreadExec** - This function is called by the Resume function to add this process into the subagent's internal table and update the binding vector and binding file. A thread is used so that EcAgManager doesn't have to wait for the Resume to finish. This function will also notify HP Open View that the process has resumed.

Arguments:pthread\_addr\_t pthread

Return Type:static pthread\_addr\_t

Privilege:Private

PDL:pthread\_addr\_t MsAgRegistry::ResumeThreadExec (pthread\_addr\_t Arguments)

```
// {
```

```
//Cast void * argument to resume parameter.
```

```
//MsTAgResumeParam *pResume = (MsTAgResumeParam *) Arguments;
```

```
//
```

```

//Attempt to bind to the manager process.
//status = pResume->pMgrHdle->Bind();
//If binding is successful, save the binding information.
//Add this process and it's metric information to the corresponding tables.
//status = pResume->pthis->AddEntryToTables (pResume->pMgrHdle);
//If previous status is OK, create an event to notify HPOV that this application has
//resumed.
//}

```

**SaveMgmtHandle** - This function is used while suspending a process. It saves the suspended entry from the binding vector file to a suspended vector file.

Arguments:MsAgMgmtHandle \*pHdle

Return Type:EcUtStatus

Privilege:Private

PDL:EcUtStatus MsAgRegistry::SaveMgmtHandle ( MsAgMgmtHandle \*pHdle )

```

// {
//   EcUtStatus status;
//   insert the entry into suspended vector
//   pSuspendedVector->insert(pHdle);
//   remove the entry from the binding vector
//   RemoveMgmtHandle (pHdle);
//   return status
// }

```

## Start

Arguments:EcTInt nNewAppID, EcTInt nNewProg, EcTInt nNewProcID, EcTInt  
nNewInstID, MsAgStringBuff sNewMode, EcTULongInt nNewRWTime, DCEObjRefT  
\*pNewDCbjRef

Return Type:EcTVoid

Privilege:Public

## StartThreadExec

Arguments:pthread\_addr\_t pThread

Return Type:static pthread\_addr\_t

Privilege:Private

## Stop

Arguments:EcTULongInt nNewAppID, EcTULongInt nNewProg, EcTULongInt  
nNewProcID, EcTULongInt nNewInstID, EcTULongInt nNewShutSecs

## StopThreadExec

Arguments:pthread\_addr\_t pThread

Return Type:static pthread\_addr\_t

Privilege:Private

### Suspend

Arguments:EcTULongInt nNewAppID, EcTULongInt nNewProg, EcTULongInt nNewProcID, EcTULongInt nNewInstID, EcTULongInt nNewShutSec

```
PDL:EcTVoid MsAgRegistry::Suspend ( )
{//
// create a MsEAgSuspendParam object to send when to start the stop monitoring
// thread.
//
// Create thread to save the internal tables for this process.
// DCEPthread *pThread = new DCEPthread ( SuspendThread, (pthread_addr_t) (void *)
pSuspendParam);
// delete the thread
// }
```

### SuspendThreadExec

Arguments:pthread\_addr\_t pthread

Return Type:static pthread\_addr\_t

Privilege:Private

```
PDL:pthread_addr_t MsAgRegistry::SuspendThreadExec (pthread_addr_t Arguments)
//{
//cast void * arguments to stop parameter structure
//MsTAgSuspendParam *pSuspend = (MsTAgSuspendParam *) Arguments;
//EcUtStatus status;
//
//save the table entry for later use during resume function.
//Remove the table entries first so that no other thread will attempt to access this binding
//handle through table entry. Remove all associated entries with this process and instance.
//
//Save a copy of the binding handle in the suspended vector and remove it from the
//current vector.
//Delete the entry in the vector that has the same attributes as MgmtObject.
//Send an event to HPOV to notify that this process has been suspended.
// Delete the suspend struct parameter
//Delete pSuspend;
//}
```

### ~MsAgRegistry -

Arguments:

Return Type:Void

Privilege:Public

## **Associations:**

The MsAgRegistry class has associations with the following classes:

Class: MsAgTblMgr accessedby  
Class: EcAgManager notifies  
Class: MsAgMgmtBindingVector uses  
MsAgSubAgent (Aggregation)

### **4.1.3.46 MsAgSNMPTbl Class**

Parent Class:Not Applicable

Public:No

Distributed Object:No

Purpose and Description:

he MsAgSNMPTbl represents a physical SNMP table in a MIB. The table supports full SNMP query capabilities (including getNext and getLocate). The class is basically a RWSortedVector of MsAgTblEntries. The enties are sorted in the vector by their corresponding index.

## **Attributes:**

**nLastIndexID** - This attribute contains the last index id.

**pLock** - This attribute represents entrance/exit class lock.

## **Operations:**

**Get** - This method will get element with specified index.

Arguments:EcTInt nTblIndex

PDL:MsAgTblEntry\* MsAgSNMPTbl::Get ( EcTInt nTblIndex )

{

create a table entry containing the Index of the entry to be gotten

MsAgTblEntry \*pEntry = NULL;

MsAgTblEntry Key ( nTblIndex, 0 );

UINT32 nIndex;

find the entry in the sorted vector

if ( ( nIndex = RWSortedVector::index ( &Key ) ) != RW\_NPOS )

pEntry = (MsAgTblEntry\*)(\*this)[nIndex];

return the entry

return pEntry;

}

**GetFirst** - This method will get the first element in the table.

Arguments:

PDL:MsAgTblEntry\* MsAgSNMPTbl::GetFirst ( )

{

    MsAgTblEntry \*pEntry = NULL;

    RWCollectableInt \*pIndex;

    get the first item in the sorted vector

    pEntry = (MsAgTblEntry\*)RWSortedVector::first ();

    return the first item

    return pEntry;

}

**GetNext** - This method will get element that procedes index value passed.

Arguments:EctInt nTblIndex

PDL:MsAgTblEntry\* MsAgSNMPTbl::GetNext ( EctInt nTblIndex )

{

    create a table entry containing the Index of the entry to be  
    gotten

    MsAgTblEntry \*pEntry = NULL;

    MsAgTblEntry Key ( nTblIndex, 0 );

    UINT32 nIndex;

    get the index into the sorted vector of the item with index nTblIndex

    if ( ( nIndex = RWSortedVector::index ( &Key ) ) != RW\_NPOS ) {

        try {

            if ( (nIndex + 1) != entries () ) {

                get the next item in the sorted vector

                pEntry = (MsAgTblEntry\*)(\*this)[nIndex+1];

            }

        }

        caught when no more items in table

        catch ( RWBoundsErr Error ) {

            pEntry = NULL;

        }

    }

```
return the entry  
return pEntry;  
}
```

**Lock** - This method locks the table.

Arguments:

```
PDL:EcTVoid MsAgTblEntry::Lock ()  
{  
    lock the mutex  
    pLock->Lock ();  
}
```

**MsAgSNMPTbl** - This method represents the default constructor.

Arguments:

```
PDL:MsAgSNMPTbl::MsAgSNMPTbl ()  
{  
    start LastIndex at 0  
    nLastIndexID = 0;  
  
    allocate space for the mutex  
    pLock      = new DCEPthreadMutex;  
}
```

## Remove

Arguments:Ec TInt nTblIndex

**TryLock** - This method tries to lock the table.

Arguments:

```
PDL:Ec TInt MsAgTblEntry::TryLock ()  
{  
    try to lock the mutex  
    pLock->Try ();  
}
```

**UnLock** - This method unlocks the table.

Arguments:

```
PDL:EcTVoid MsAgTblEntry::UnLock ()  
{  
    unlock the mutex  
    pLock->UnLock ();
```

```
}
```

**findInstanceID** - This method returns an entry with the specified InstanceID if it exists.

Arguments:EctInt nTblIndex

```
PDL:MsAgTblEntry* MsAgSNMPTbl::findInstanceID (nTblIndex)
```

```
{
```

create a table entry containing the Index of the entry to be  
found

```
MsAgTblEntry *pEntry = NULL;
```

```
MsAgTblEntry Key ( nIndex, 0 );
```

ask the sorted vector for the index of the item (if it exists)  
and get the entry

```
if ( ( nIndex = RWSortedVector::index ( &Key ) ) != RW_NPOS )
```

```
pEntry = (MsAgTblEntry*)(*this)[nIndex];
```

return a pointer to the entry

```
return pEntry;
```

```
}
```

**insert** - This method will add an snmp table entry; uses lastindexid as tableindex.

Arguments:RWCollectable \*c

```
PDL:MsAgTblEntry* MsAgSNMPTbl::insert ( RWCollectable *c )
```

```
{
```

cast the collectable into a table entry

```
MsAgTblEntry *pEntry = (MsAgTblEntry*)c;
```

set the index of the table entry, and increment the counter

```
pEntry->nTblIndex = nLastIndexID++;
```

insert the entry into the sorted vector

```
RWSortedVector::insert ( c );
```

return the inserted entry

```
return c;
```

```
}
```

**isEmpty** - This method returns true if no entries are in the SNMP table.

Arguments:

```
PDL:RWBoolean * MsAgSNMPTbl::isEmpty ( ) const
```

```
{
```

```
RWBoolean b;
```

```
ask Sorted Vector if it is empty  
b = RWSortedVector::isEmpty ();  
  
return true is vector is empty  
return b;  
}
```

**~MsAgSNMPTbl** - This method represents the default destructor.

Arguments:

```
PDL::MsAgSNMPTbl::~MsAgSNMPTbl ()  
{  
    free the memory taken by the mutex  
    delete pLock  
}
```

### Associations:

The MsAgSNMPTbl class has associations with the following classes:

MsAgTblMgr (Aggregation)

#### 4.1.3.47 MsAgScheduleEntry Class

Parent Class:Not Applicable

Public:No

Distributed Object:No

Purpose and Description:

This class represents one entry in the schedule that contains the function pointer, the time the function is supposed to be executed, arguments for the function in question, and the index of the entry.

### Attributes:

None

### Operations:

**GetIndex** - returns index

Arguments:

**GetInterval** - returns interval

Arguments:

**GetRepeatOn** - returns repeat on flag.

Arguments:

**MsAgScheduleEntry** - default constructor assigns values

Arguments:DCEPthreadProc pNewFunction, DCEPthreadParam pNewParameters, RWTime rtNewExecutionTime, EcTInt nNewInterval, EcTInt n newIndex., EcTBoolean newRepeatScheduling=EcDtrue

#### **Associations:**

The MsAgScheduleEntry class has associations with the following classes:

Class: MsAgScheduler uses

#### **4.1.3.48 MsAgScheduler Class**

Parent Class:Not Applicable

Public:No

Distributed Object:No

Purpose and Description:

The scheduler provides the facility to execute a function after constant specified time intervals. The scheduler may contain many entries. A user can ask the scheduler to insert and/or remove an entry. This class is generic to allow other groups to use it instead of designing their own scheduler.

#### **Attributes:**

**keepRunning** - used by destructor to tell schedule thread to end scheduling.

**mutex** - mutex needed by condition thread

**nIndex**

**nInterval**

**nLastIndex** - last index for schedule entries

**pCondThread** - condition thread

**pFunction**

**pLock** - scheduler mutex

**pParameters**

**pThread** - schedule thread.

**repeatOn**

**rtExecutionTime**

**scheduleList** - list of schedule entries

**wakeUpEarly** - needed to tell a thread to wake up early.

### **Operations:**

**Delete** - deletes entry from schedule.

Arguments:EcTInt nIndex

**GetExecutionTime**

Arguments:

**GetFunctionPointer**

Arguments:

**GetParamterPointer**

Arguments:

**ModifyEntry** - This method inserts an entry into the schedule. It returns the index which can be used to remove the entry before the method is executed. If interval is -1 or less then execute once. If interval is 0 then turn scheduling off. If interval is 1 or bigger then update interval. If index is -1 or less then add entry to schedule.

Arguments:EcUtStatus &status, DCEPthreadProc pFunction, DCEPthreadParam pParameters, EcTInt nInterval=-1, EcTInt nIndex=-1, EcTBoolean repeatOn=EcDTrue

**MsAgScheduler** - default constructor

Arguments:

**SchdThreadExec** - waits then executes scheduled function and finally inserts another entry into the schedule with the same index.

Arguments:pthread\_addr\_t thrd

**compareTo**

Arguments:const RWCollectable \*c

**isEqual**

Arguments:const RWCollectable \*c

### **~MsAgScheduleEntry**

Arguments:

### **~MsAgScheduler - default destructor**

Arguments:

### **Associations:**

The MsAgScheduler class has associations with the following classes:

Class: MsAgSubAgent usedby

Class: MsAgScheduleEntry uses

### **4.1.3.49 MsAgSnmpPdu Class**

Parent Class:Not Applicable

Public:No

Distributed Object:No

Purpose and Description:

This file contains the header file information for the MsAgSnmpPdu. This class is a wrapper for HPOV's pdu class library.

### **Attributes:**

**pPdu** - pointer to the OV pdu object

### **Operations:**

#### **Add - insert a variable binding**

Arguments:MsTAgSnmpObject \*pObject, EcTVoid \*pValue, EcTInt nValueLength

#### **Add - insert a variable binding**

Arguments:MsTAgSnmpObject \*pObject, EcTInt nValue, EcTInt nValueLength

#### **MsAgSnmpPdu - default constructor**

Arguments:EcTInt nPduType, EcTInt nTrapNumber

#### **Send - process the pdu (i.e. send as trap, ... )**

Arguments:OVsnmpSession \*pSession

#### **operator**

Arguments:

### **Associations:**

The MsAgSnmpPdu class has associations with the following classes:

Class: MsAgDeputy usedby

#### 4.1.3.50 MsAgStaticBuffer Class

Parent Class:Not Applicable

Public:No

Distributed Object:No

Purpose and Description:

The StaticBuffer is used to store information from the MsAgTblEntry. It's created once by the MsAgSubAgent and is used globally. The StaticBuffer needs to be global in order for Peer to access it. The StaticBuffer holds 2 types of data. The first type is a large data buffer. The second is an array of small buffers, where ASCII-Z strings are stored.

#### Attributes:

**nBufSize** - This attribute represents the size of buffer (in bytes).

**nMaxStringSize** - This attribute represents maximum length of a string in the static buffer.

**nNumStrings** - This attribute represents the number of strings spaces allocated.

**pBuffer** - This attribute represents a pointer to byte buffer.

**szStrings** - This attribute represents a pointer to array of character strings.

#### Operations:

**Get** - This method returns a pointer to the data in the static buffer.

Arguments:

**GetString** - This method returns a pointer to the string in the string buffer indexed by nStringNumber.

Arguments:EctInt nStringNumber

**MsAgStaticBuffer** - This method represents the default constructor. Creates a MsAgStaticBuffer object with the specified number of bytes, and specified number of C-Strings.

Arguments:EctInt nNewBufSize, EctInt nNewNumStrings, EctInt nNewMaxStringSize=255

#### Set

Arguments:EctPptr pNewBuffer, EctInt nNumBytes

### **SetString**

Arguments:EcTInt nStringNumber, RWCString &rsString

**~MsAgStaticBuffer** - This method represents the default destructor.

Arguments:

### **Associations:**

The MsAgStaticBuffer class has associations with the following classes:

Class: MsAgTblEntry uses

MsAgSubAgent (Aggregation)

### **4.1.3.51 MsAgSubAgent Class**

Parent Class:Not Applicable

Public:Yes

Distributed Object:Yes

Purpose and Description:

This managed object class supports SNMP MIB extensions. It receives requests from the master agent. Based on Get or Set requests, it performs the retrieval or set functions onto resource or resource managers using available API. This object will instantiate another object MsAgMonitor to perform local polling on resources on the host.

### **Attributes:**

**nDiscoverIndex** - discover index into scheduler

**nDiscoverInterval** - discover interval

**nPerformanceIndex** - performance index into scheduler

**nPerformanceInterval** - performance interval

**nStatusIndex** - status index into scheduler

**nStatusInterval** - status polling interval

**pDeputyGate**

**pDiscoverer** - This attribute represents a pointer to a discoverer.

**pLock** - DCE thread mutex

**pMonitor** - This attribute represents a pointer to a monitor.

**pPortMonitor** - This attribute represents a pointer to a port monitor.

**pRegistry** - registry pointer

**pScheduler** - scheduler pointer

**pSubAgentConfig**

**pTblMgr** - This attribute represents a pointer to a table manager.

### **Operations:**

**ErrorHandling** - performs error handling for PerformTask

Arguments:MsTAgSubAgentTaskType taskType, EcTInt nNewInterval

**MsAgSubAgent** - This method represents the constructor of the object.

Arguments:

**PerformTask** - Removes entry from schedule which would poll using the old interval, and insert an entry into the schedule with the new interval. OR Stop repeating specified task. OR tells discoverer to discover all applications or tell monitor to poll performance/status immediately and one time only OR tell monitor to poll performance after specified time interval and only once

Arguments:MsTAgSubAgentTaskType taskType, EcTInt nNewInterval=-1

**SchdThreadExec** - This method spawns a DCE thread.

Arguments:pthread\_addr\_t pThread

**~MsAgSubAgent** - This method represents the destructor of the object.

Arguments:

### **Associations:**

The MsAgSubAgent class has associations with the following classes:

Class: MsAgAgent communicateswith

Class: MsAgScheduler usedby

Class: EcAgProxy uses

Class: MsAgAppMIB uses

#### **4.1.3.52 MsAgSubAgentConfig Class**

Parent Class:Not Applicable

Public:No

Distributed Object:No

Purpose and Description:

This class contains configuration information for the SubAgent.

### Attributes:

**nEventScope** - This attribute represents the scope of the event for forwarding (e.g. order entry).

**nLocalPerfPollInterval** - This attribute represents the time interval of the polling of performance metrics on this host.

**nLocalPollInterval** - This attribute represents the time interval of the polling of this host.

**nLocalPollScope** - This attribute represents the scope of the polling on the host.

**nLogLevel** - This attribute represents the level of event for logging (e.g. not logging).

**nMSSLogSizeThreshold** - attribute representing the threshold mark for the MSS log.

**nRetry** - This attribute represents the number of times to retry to bind the application.

**nRetrySleep** - This attribute represents the time to sleep before it retries again.

**nUpTime** - This attribute represents the value of sysUpTime at the time this agent was last initialized.

**pLock** - This attribute represents a DCE thread mutex.

**rsDeputyCDSName** - attribute representing the Deputy name

**rsMode**

### Operations:

**GetDeputyCDSName** - returns deputy cds name

Arguments:

**GetEventScope** - This method gets the event scope.

Arguments:

**GetLocalPerfPollInterval** - This method gets the performance polling interval.

Arguments:

**GetLocalPollInterval** - This method gets the polling interval.

Arguments:

**GetLocalPollScope** - This method gets the polling scope.

Arguments:

**GetMSSLogSizeThreshold**

Arguments:

**GetMaxMSSLogSize**

Arguments:

**GetMode** - This method gets the mode.

Arguments:

**GetRetry** - This method gets the number of time to retry.

Arguments:

**GetRetrySleep** - This method gets the time to sleep.

Arguments:

**GetUpTime** - This method gets the uptime.

Arguments:

**Lock** - This method locks subagent's table.

Arguments:

**MsAgSubAgentConfig** - This method represents the default constructor.

Arguments:

**SetDeputyCDSName** - sets deputy cds name

Arguments:RWCString rsNewDeputyCDSName

**SetEventScope** - This method sets the event scope.

Arguments:EcTInt nNewEventScope

**SetLocalPerfPollInterval** - This method sets the performance polling interval.

Arguments:EcTInt nNewLocalPerfPollInterval

**SetLocalPollInterval** - This method sets the polling interval.

Arguments:EcTInt nNewLocalPollInterval

**SetLocalPollScope** - This method sets the polling scope.

Arguments:EcTInt nNewLocalPollScope

**SetMSSLogSizeThreshold**

Arguments:EcTInt nNewLogSizeThreshold

**SetMode**

Arguments:RWCString rsNewMode

**SetRetry** - This method sets the number of times to retry.

Arguments:EcTInt nNewRetry

**SetRetrySleep** - This method sets the time to sleep before retrying.

Arguments:EcTInt nNewRetrySleep

**SetUpTime** - This method sets the uptime.

Arguments:EcTInt nNewUpTime

**UnLock** - This method unlLocks subagent's table.

Arguments:

**~MsAgSubAgentConfig** - This method represents the default destructor.

Arguments:

**Associations:**

The MsAgSubAgentConfig class has associations with the following classes:

Class: MsAgEventMgr uses  
MsAgSubAgent (Aggregation)

#### 4.1.3.53 MsAgTblEntry Class

Parent Class:Not Applicable

Public:No

Distributed Object:No

Purpose and Description:

This is an abstract class representing an entry in an SNMP table. Each TblEntry corresponds to a single row in the table. Each entry has a unique index (used by the SNMP protocol), an instance ID which specifies the run instance the entry belongs to, and a parent ID, which specifies its parent TblEntry.

**Attributes:**

**nIndex** - This attribute represents the SNMP table index.

**nInstanceID** - This attribute represents the instance number of the table entry.

**nParentID** - This attribute represents the ID of parent.

### Operations:

**GetIndex** - This method gets the index of the table entry.

Arguments:

```
PDL:EcTInt MsAgTblEntry::GetIndex ()  
{  
    return nIndex;  
}
```

**GetInstanceID** - This method gets the instance ID of the table entry.

Arguments:

```
PDL:EcTInt MsAgTblEntry::GetInstanceID ()  
{  
    Return the instance ID  
    return nInstanceID;  
}
```

**GetParentID** - This method gets the parent ID of the table entry.

Arguments:

```
PDL:EcTInt MsAgTblEntry::GetParentID ()  
{  
    Return the ParentID  
    return nParentID;  
}
```

### GetValues

Arguments:MsAgStaticBuffer \*pBuffer, EcUtStatus &status

**MsAgTblEntry** - This method represents the default constructor.

Arguments:

```
PDL:MsAgTblEntry::MsAgTblEntry ()  
{  
    Initialize the attributes  
    nIndex      = 0;  
    nParentIndex = 0;  
    nInstanceID = 0;
```

set the time one year backwards to guarantee the data will be updated

```
#seconds * #minutes * #hours * #days  
rtLastModified -= 60 * 60 * 24 * 365;  
}
```

**MsAgTblEntry** - constructor that assigns values  
Arguments:EcTInt nNewParentID, EcTInt nNewID

### **MsAgTblEntry**

Arguments:EcTInt nNewIndex, EcTInt nSomething, EcTInt nSomethingElse

**SetIndex** - This method sets the index of the table entry.

```
Arguments:EcTInt nNewIndex  
PDL:EcTInt MsAgTblEntry::SetIndex ( EcTInt nNewIndex )  
{  
    Set the Index ID  
    return ( nIndex = nNewTblIndex );  
}
```

**SetInstanceID** - This method sets the instance ID of the table entry.

```
Arguments:EcTInt nNewInstanceID  
PDL:EcTInt MsAgTblEntry::SetInstanceID ( EcTInt nNewInstanceID )  
{  
    Set the instance ID  
    return ( nInstanceID = nNewInstanceID );  
}
```

**SetParentID** - This method sets the parent ID of the table entry.

```
Arguments:EcTInt nNewParentID  
PDL:EcTInt MsAgTblEntry::SetParentID ( EcTInt nNewParentID )  
{  
    return ( nParentID = nNewParentID );  
}
```

### **SetValues**

Arguments:MsTAgSNMPEEntry \*pData, EcTInt nAttribID

**compareTo** - This method compares the table entry againts another (used by RW).

```
Arguments:const RWCollectable *c  
PDL:virtual EcTInt MsAgTblEntry::compareTo ( const RWCollectable *c ) const;  
{  
    cast c as a TblEntry  
    MsAgTblEntry *p2 = (MsAgTblEntry*)c;  
  
    return the difference between the two TblEntries Indexes
```

```
    return ( nIndex - p2->nIndex );
}
```

**isEqual** - This method returns true if the table entry is equal to the entry passed in (used by RW).

Arguments:const RWCollectable \*c  
PDL:virtual RWBoolean MsAgTblEntry::isEqual ( const RWCollectable\* ) const;  
{  
 return true if the two TblEntry Indexes are equal  
 return ( nIndex == ((MsAgTblEntry\*)(c))->nIndex );  
}

**~MsAgTblEntry** - This method represents the default destructor.

Arguments:  
PDL:MsAgTblEntry::~MsAgTblEntry ()  
{  
}

### **Associations:**

The MsAgTblEntry class has associations with the following classes:

Class: MsAgStaticBuffer uses  
MsAgSNMPTbl (Aggregation)

### **4.1.3.54 MsAgTblMgr Class**

Parent Class:Not Applicable

Public:No

Distributed Object:No

Purpose and Description:

The TableManager handles tables. Almost all of the managers/monitors use it to access a table.

### **Attributes:**

**pTables** - This attribute represents an array of all the table entries.

### **Operations:**

**AccessTbl** - This method accesses the specified SNMP table.

Arguments:MsTAgSNMPTblID tblID

**FoundMatch** - This method compares a metric table entry and a mgmt handle.  
Arguments:MsAgMgmtHandle handle, MsgTeIndx metricTblEntry, MsTAgSNMPTblID tblID

**.LogError** - logs an error message.  
Arguments:

**MsAgTblMgr** - This method represents the default constructor.  
Arguments:

**RemoveDyAppTblEntries** - removes entries from MsEAgDyAppTbl table  
Arguments:EcTInt nAppID, RWCString rsMode, EcTInt nInstanceID

**RemoveDyProcTblEntries** - removes entries from MsEAgDyProcTbl table  
Arguments:RWOrdered linkedList, RWCString rsMode, EcTInt nInstanceID

**RemoveDyProgTblEntries** - removes entries from MsEAgDyProgTbl table  
Arguments:EcTInt nAppID, EcTInt nProgID, RWCString rsMode, EcTInt nInstanceID

**RemoveDynAppRef** - Only the dynamic information about the specified application with the corresponding entry ID and instance ID is removed.  
Arguments:EcTInt nAppID, RWCString rsMode, EcTInt nInstanceID

**RemoveDynProcRef** - This method remove information on the process level.  
Arguments:EcTInt nProcID

**RemoveDynProgRef** - Only the dynamic information about the specified application with the corresponding entry ID and instance ID is removed.  
Arguments:EcTInt nProgID, RWCString rsMode, EcTInt nInstanceID

**RemoveMetricRef** - This method removes all metric table entries which have corresponding app, prog, proc & instance IDs.  
Arguments:EcTInt nAppID, EcTInt nProgID, EcTInt nProcID, EcTInt nInstID

**RemoveMetricRef** - this method removes metric references.  
Arguments:RWOrdered linkedList, MsTAgSNMPTblID tblID

**RemoveRef** - This method removes DYNAMIC information in table entries associated with an application, program, and/or process. The instance ID may only be left blank if the level refers to the process.  
Arguments:EcTAgMgmtLevel level, EcTInt nLevelID, EcTInt nInstanceID = -1

**RemoveRef** - This method removes both STATIC and DYNAMIC information in table entries associated with an application and/or program. The level should only be app or prog

(not proc). If level 'proc' is used an error will be returned.

Arguments:EcTAgMgmtLevel level, EcTInt nLevelID, RWCString rsMode

**RemoveStAppTblEntries** - removes entries from MsAgTeStApp table

Arguments:EcTInt nAppID, RWCString rsMode

**RemoveStDynAppRef** - This method removes all information on the application level.

Arguments:EcTInt nAppID, RWCString rsMode

**RemoveStDynProgRef** - This method remove information on the program level.

Arguments:EcTInt nProgID, RWCString rsMode

**RemoveStProgTblEntries** - removes entries from MsAgTeStProg table

Arguments:EcTInt nAppID, EcTInt nProgID, RWCString rsMode

**~MsAgTblMgr** - This method represents the default destructor.

Arguments:

### **Associations:**

The MsAgTblMgr class has associations with the following classes:

Class: MsAgPortMonitor accessedby

Class: MsAgRegistry accessedby

MsAgSubAgent (Aggregation)

#### **4.1.3.55 MsAgTeConfig Class**

Parent Class:MsAgTeIndx

Public:No

Distributed Object:No

Purpose and Description:

This class is a table that stores information about configuration data.

### **Attributes:**

**rsValue** - This attribute contains the value.

### **Operations:**

**GetValue** - returns value

Arguments:

**GetValues**

Arguments:MsAgStaticBuffer \*pBuffer, EcUtStatus &status

**MsAgTeConfig**

Arguments:EcTInt nNewParentID,EcTInt nNewID, EcTAgMgmtLevel newLevel

**SetLevel** - Sets level attribute

Arguments:EcTAgMgmtLevel newLevel

**SetValue** - sets value

Arguments:RWCString &rsnewValue

**SetValues**

Arguments:MsTAgSNMPEntry \*pData, EcTInt nAttribID

**UpdateValues** - This method internally updates the values of any dynamic attributes.

Arguments:

**~MsAgTeConfig** - This method represents the destructor.

Arguments:

**Associations:**

The MsAgTeConfig class has associations with the following classes:

None

**4.1.3.56 MsAgTeDyProc Class**

Parent Class:MsAgTeDynamic

**Attributes:****nGID****nPPID****nPriority****nProcFile /\*SUN\*/****nUID****procInfo****rsServObjID**

## **rsTTY**

**rtTimeToStop** - This attribute represents the time to stop.

### **Operations:**

#### **GetGID**

Arguments:

#### **GetPPID**

Arguments:

#### **GetPriority**

Arguments:

**GetServerObjID** - This method gets the server object ID.

Arguments:

#### **GetTImeToStop**

Arguments:

#### **GetTTY**

Arguments:

#### **GetUID**

Arguments:

**GetValues** - This method gets values. return pointer to SNMP readable data.

Arguments:MsAgStaticBuffer \*pBuffer, EcUtStatus &status

**MsAgTeDyProc** - This method represents the constructor.

Arguments:EcTInt nNewParentID, EcTInt nNewID

#### **OpenProcFile /\*SUN\*/**

Arguments:

#### **SetGID**

Arguments:

#### **SetPPID**

Arguments:

#### **SetPriority**

Arguments:

**SetServerObjID** - This method sets the server object ID  
Arguments:

#### **SetTTY**

Arguments:

**SetTimeToStop** - This method sets the time to stop.  
Arguments: RWTime rtNewTimeToStop

#### **SetUID**

Arguments:

**SetValues** - sets values.

Arguments: MsTAgSNMPEntry \*pData, EcTInt nAttribID

#### **~MsAgTeDyProc**

Arguments:

### **Associations:**

The MsAgTeDyProc class has associations with the following classes:

None

### **4.1.3.57 MsAgTeDynamic Class**

Parent Class: MsAgTeMdMg

Public: No

Distributed Object: No

Purpose and Description:

This is an abstract class for dynamic table entries

### **Attributes:**

**nUpTime** - uptime of the entry

### **Operations:**

**GetUpTime** - returns uptime

Arguments:

**GetValues** - gets all values, stores them in a structure and returns a pointer to SNMP readable data.

Arguments: MsAgStaticBuffer \*pBuffer, EcUtStatus &status

**MsAgTeDynamic** - default constructor  
Arguments:EcTInt nNewParentID, EcTInt nNewID

**SetUpTime** - sets uptime  
Arguments:EcTInt nUpTime

**SetValues** - Sets the values of the table entry  
Arguments:MsTAgSNMPEntry \*pData, EcTInt nAttribID

**isEqual** - returns true if the table entry is equal to the entry passed in (used by RW)  
Arguments:const RWCollectable\* c

**~MsAgTeDynamic** - default destructor  
Arguments:

#### **Associations:**

The MsAgTeDynamic class has associations with the following classes:  
None

#### **4.1.3.58 MsAgTeFault Class**

Parent Class:MsAgTeIndx  
Public:No  
Distributed Object:No  
Purpose and Description:  
This class is a table that stores information about fault data.

#### **Attributes:**

**nValue** - This attribute contains the value.

#### **Operations:**

**GetValue** - returns value  
Arguments:  
**GetValues**  
Arguments:MsAgStaticBuffer \*pBuffer, EcUtStatus &status

**MsAgTeFault**  
Arguments:EcTInt nNewParentID, EcTInt nNewID, EcTAgMgmtLevel newLevel

**SetLevel** - sets mgmt level  
Arguments:EcTAgMgmtLevel newLevel

**SetValue** - sets value  
Arguments:EcTInt nnewValue

**SetValues**  
Arguments:MsTAgSNMPEntry \*pData, EcTInt nAttribID

**UpdateValues** - This method internally updates the values of any dynamic attributes.  
Arguments:

**~MsAgTeFault** - This method represents the destructor.  
Arguments:

### **Associations:**

The MsAgTeFault class has associations with the following classes:  
None

#### **4.1.3.59 MsAgTeIndx Class**

Parent Class:MsAgTblEntry  
Public:No  
Distributed Object:No  
Purpose and Description:  
This class is an abstract class which contains a MetricIndex and a BindingIndex.

### **Attributes:**

**indxList** - This attribute contains a pointer to the handle which contains a pointer to the EcAgManager object and the index to the actual metric function.

**level** - mgmt level.

**rsType** - type attribute

**rtLastModified** - last modified attribute

### **Operations:**

**GetIndx** - gets entry from ordered list.  
Arguments:EcTInt nEntry

**GetLevel** - returns mgmt level

Arguments:

**GetType** - returns type

Arguments:

**Insert** - inserts entry into the ordered list.

Arguments:MsAgMgmtIdx \*newIndx

**MsAgTeIdx**

Arguments:EcTInt nNewParentID, EcTInt nNewID

**NeedsRefresh** - True if data has become stale

Arguments:

**NumEntries** - returns the number of entries in MsAgMgmtIdx.

Arguments:

**RemoveMgmtIndx** - removes entry from ordered list.

Arguments:EcTInt nAppID, EcTInt nProgID, EcTInt nProcID, EcTInt nInstID

**ResetLastModified** - Sets the time last modified to the current time

Arguments:

**SetLevel** - sets mgmt level

Arguments:EcTAgMgmtLevel newLevel

**SetType** - sets type

Arguments:RWCString &rsNewType

**UpdateValues** - Internally updates the values of any dynamic attributes

Arguments:

**isEqual** - returns true if the table entry is equal to the entry passed in (used by RW)

Arguments:const RWCollectable\* c

**~MsAgTeIdx** - This method represents the destructor.

Arguments:

## Associations:

The MsAgTeIdx class has associations with the following classes:

None

#### **4.1.3.60 MsAgTeMdMg Class**

Parent Class:MsAgTblEntry

Public:No

Distributed Object:No

Purpose and Description:

This class is a table that stores information about mode of an application/program/process.

#### **Attributes:**

**rsMode**

#### **Operations:**

**GetMode**

Arguments:

**MsAgTeMdMg**

Arguments:EcTInt nNewParentID, EcTInt nNewID

**SetMode**

Arguments:RWCString rsNewMode

**~MsAgTeMdMg**

Arguments:

#### **Associations:**

The MsAgTeMdMg class has associations with the following classes:

None

#### **4.1.3.61 MsAgTePerf Class**

Parent Class:MsAgTeIndx

Public:No

Distributed Object:No

Purpose and Description:

This class is a table that stores information about performance data.

#### **Attributes:**

**lastState** - last state of threshold

**nFaultThreshold** - This attribute contains the fault threshold.

**nMaxThreshold** - This attribute contains the maximum threshold.

**nMinThreshold** - This attribute contains the minimum threshold.

**nRearmFaultThreshold** - This attribute contains the fault rearm threshold.

**nRearmMaxThreshold** - This attribute contains the maximum rearm threshold.

**nRearmMinThreshold** - This attribute contains the minimum rearm threshold.

**nValue** - This attribute contains the value.

### Operations:

**GetFaultThreshold** - returns fault threshold

Arguments:

**GetLastState** - returns last state attribute

Arguments:

**GetMaxThreshold** - returns max threshold

Arguments:

**GetMinThreshold** - returns min threshold

Arguments:

**GetRearmFaultThreshold** - returns rearm fault threshold

Arguments:

**GetRearmMaxThreshold** - returns rearm max threshold

Arguments:

**GetRearmMinThreshold** - returns rearm min threshold

Arguments:

**GetValue** - returns value attribute

Arguments:

**GetValues**

Arguments:MsAgStaticBuffer \*pBuffer, EcUtStatus &status

**MsAgTePerf**

Arguments:EcTInt nNewParentID, EcTInt nNewID, EcTAgMgmtLevel newLevel

**SetFaultThreshold** - sets fault threshold  
Arguments:EcTInt nNewFaultThreshold

**SetLastState** - sets last state attribute  
Arguments:MsTAgThresholdState newLastState

**SetLevel** - sets level attribute  
Arguments:EcTAgMgmtLevel level

**SetMaxThreshold** - sets max threshold  
Arguments:EcTInt nNewMAxThreshold

**SetMinThreshold** - sets min threshold  
Arguments:EcTInt nNewMinThreshold

**SetRearmFaultThreshold** - sets rearm fault threshold  
Arguments:EcTInt nNewRearmFaultThreshold

**SetRearmMaxThreshold** - sets rearm max threshold  
Arguments:EcTInt nNewRearmMaxThreshold

**SetRearmMinThreshold** - sets rearm min threshold  
Arguments:EcTInt nNewRearmMinThreshold

**SetValue** - sets value attribute  
Arguments:EcTInt nnewValue

**SetValues**  
Arguments:MsTAgSNMPEntry \*pData, EcTInt nAttribID

**UpdateValues** - This method internally updates the values of any dynamic attributes.  
Arguments:

**~MsAgTePerf** - This method represents the destructor.  
Arguments:

### **Associations:**

The MsAgTePerf class has associations with the following classes:  
None

#### **4.1.3.62 MsAgTeStApp Class**

Parent Class:MsAgTeVerExec

Public:No

Distributed Object:No

Purpose and Description:

This class contains static application information.

#### **Attributes:**

**nEventLevel**

**nLogLevel** - log level.

**nLogMax**

**rsLanguage**

**rsLogPath**

#### **Operations:**

**GetEventLevel**

Arguments:

**GetLanguage**

Arguments:

**GetLogLevel** - returns log level

Arguments:

**GetLogMax**

Arguments:

**GetLogPath**

Arguments:

**GetValues**

Arguments:MsAgStaticBuffer \*pBuffer, EcUtStatus &status

**MsAgTeStApp**

Arguments:EcTInt nNewParentID, EcTInt nNewID

**SetEventLevel**

Arguments:EcTInt

**SetLanguage**

Arguments:RWCString rsNewLanguage

**SetLogLevel** - sets log level

Arguments:EcTInt nNewLogLevel

**SetLogMax**

Arguments:EcTInt

**SetLogPath**

Arguments:RWCString

**SetValues** - sets values

Arguments:MsTAgSNMPEntry \*pData, EcTInt nAttribID

**~MsAgTeStApp**

Arguments:

**Associations:**

The MsAgTeStApp class has associations with the following classes:

None

#### **4.1.3.63 MsAgTeStProg Class**

Parent Class:MsAgTeVerExec

**Attributes:**

**nTimeOut**

**rsAppName**

**Operations:**

**GetAppName**

Arguments:

**GetTimeOut**

Arguments:

**GetValues**

Arguments:MsAgStaticBuffer \*pBuffer, EcUtStatus &status

#### **MsAgTeStProg**

Arguments:EcTInt nNewParentID, EcTInt nNewID

#### **SetAppName**

Arguments:RWCString

#### **SetTimeOut**

Arguments:EcTInt

#### **SetValues** - sets values

Arguments:MsTAgSNMPEntry \*pData, EcTInt nAttribID

#### **~MsAgTeStProg**

Arguments:

### **Associations:**

The MsAgTeStProg class has associations with the following classes:

None

### **4.1.3.64 MsAgTeVerExec Class**

Parent Class:MsAgTeMdMg

Public:No

Distributed Object:No

Purpose and Description:

This class is a table that stores information about version execution data.

### **Attributes:**

**nNumChildren** - number of children.

**rsContact** - This attribute represents the contact.

**rsFile**

**rsMaintLevel** - This attribute represents the maintenance level.

**rsMajorVersion** - This attribute represents the major version.

**rsMinorVersion**

**rsName** - This attribute represents the name.

### **rsPath**

**rsRevision** - This attribute represents the revision.

## **Operations:**

**GetContact** - This method gets the contact.

Arguments:

```
PDL:const RWCString& MsAgTeVerExec::GetContact()
{
    // return the contact
    // return rsContact;
}
```

### **GetFile**

Arguments:

**GetInstallTime** - This method gets the installation time.

Arguments:

```
PDL:const RWCString& MsAgTeVerExec::GetInstallTime()
{
    // return the install time
    // return rsInstallTime;
}
```

**GetMaintLevel** - This method gets the maintenance level.

Arguments:

```
PDL:const RWCString& MsAgTeVerExec::GetMaintLevel()
{
    // return the maintenance level
    // return rsMaintLevel;
}
```

**GetMajorVersion** - This method gets the major version.

Arguments:

```
PDL:const RWCString& MsAgTeVerExec::GetMajorVersion()
{
    // return the major version
    // return rsMajorVersion;
}
```

```
}
```

**GetMinorVersion** - This method gets the minor version.

Arguments:

```
PDL:const RWCString& MsAgTeVerExec::GetMinorVersion()
{
// return the minor version
// return rsMinorVersion;
}
```

**GetName** - This method gets the name.

Arguments:

```
PDL:const RWCString& MsAgTeVerExec::GetName()
{
// return the Ver name
// return rsName;
}
```

**GetNumChildren** - returns number of children

Arguments:

**GetPath**

Arguments:

**GetRevision** - This method gets the revision.

Arguments:

```
PDL:const RWCString& MsAgTeVerExec::GetRevision()
{
// return the revision
// return rsRevision
}
```

**GetType** - This method gets the type.

Arguments:

PDL:

```
EcTInt MsAgTeVerExec::GetType()
{
// return the type
// return nType;
}
```

**MsAgTeVerExec**

Arguments:EcTInt nNewParentID, EcTInt nNewID

**SetContact** - This method sets the contact.

Arguments:RWCString

PDL:const RWCString& MsAgTeVerExec::GetContact()  
{  
// return the contact  
// return rsContact;  
}**SetFile**

Arguments:RWCString rsNewExecFile

**SetInstallTime** - This method sets the installation time.

Arguments:RWTime rsNewInstallTime

PDL:EcTVoid MsAgTeVerExec::SetInstallTime ( RWCString rsNewInstallTime )  
{  
// set the intall time  
// rsInstallTime = rsNewInstallTime;  
}**SetMaintLevel** - This method sets the maintenance level.

Arguments:RWCString

PDL:EcTVoid MsAgTeVerExec::SetMaintLevel( RWCString rsNewMaintLevel )  
{  
// set the maintenance level  
// rsMaintLevel = rsNewMaintLevel;  
}**SetMajorVersion** - This method sets the major version.

Arguments:RWCString rsNewMajorVersion

PDL:EcTVoid MsAgTeVerExec::SetMajorVersion ( RWCString rsNewMajorVersion )  
{  
// set the major version  
// rsMajorVersion = rsNewMajorVersion;  
}**SetMinorVersion** - This method sets the minor version.

Arguments:RWCString rsNewMinorVersion  
PDL:const RWCString& MsAgTeVerExec::GetMinorVersion()  
{  
// return the minor version  
// return rsMinorVersion;  
}

**SetName** - This method sets the name.

Arguments:RWCString rsNewName  
PDL:EcTVoid MsAgTeVerExec::SetName( RWCString rsNewName )  
{  
// set the Ver name  
// rsName = rsNewName;  
}

**SetNumChildren** - sets number of children

Arguments:EcTInt nNewNumChildren

**SetPath**

Arguments:RWCString rsNewExecPath

**SetRevision** - This method sets the revision.

Arguments:RWCString rsNewRevision  
PDL:EcTVoid MsAgTeVerExec::SetRevision( RWCString rsNewRevision )  
{  
// set the revision  
// rsRevision = rsNewRevision;  
}

**SetType** - This method sets the type.

Arguments:EcTInt nNewType  
PDL:EcTVoid MsAgTeVerExec::SetType ( EcTInt nNewType )  
{  
// set the type  
// rsType = nNewType;  
}

**~MsAgTeVerExec** - This method represents the destructor.

Arguments:  
PDL:MsAgTeVerExec::~MsAgTeVerExec ( )

```
{  
}
```

#### **Associations:**

The MsAgTeVerExec class has associations with the following classes:

None

#### **4.1.3.65 MsAgUpdateConfigMetric Class**

Parent Class:MsAgIntConfigMetric

Public:No

Distributed Object:No

Purpose and Description:

update config metric class that uses the process snapshot info.

#### **Attributes:**

**pProcInfo** - Address to the MsAgProcSShotInfo

#### **Operations:**

**MsAgUpdateConfigMetric** - Constructor for MsAgUpdateConfigMetric

Arguments:MsAgProcSShotInfo \*pNewPInfo, RWCString &rsNewType, RWCString &rsnewValue

**SetValue** - Sets value for this configuration metric and pProcInfo

Arguments:RWCString &rsnewValue

**~MsAgUpdateConfigMetric** - Destructor for MsAgUpdateConfigMetric

Arguments:

#### **Associations:**

The MsAgUpdateConfigMetric class has associations with the following classes:

Non

#### **4.1.3.66 COTSAplication Class**

Parent Class:Not Applicable

Public:No

Distributed Object:No

Purpose and Description:

Represents any ECS COTS application

**Attributes:**

None

**Operations:**

None

**Associations:**

The COTSAplication class has associations with the following classes:

Class: LogFile generates

#### **4.1.3.67 EnterpriseFramework Class**

Parent Class:Not Applicable

Public:No

Distributed Object:No

Purpose and Description:

EnterpriseFramework is the Tivoli COTS product that performs enterprise wide services: System Administration (Tivoli/Admin), Software distribution (Tivoli/Courier), performance monitoring (Tivoli/Sentry) and fault correlation (Tivoli/Enterprise Console). The framework also acts as the integrated desktop for Maintenance and Operations, integrating other administrative functions such as Sybase database administration, system backup/restore, and DCE Cell administration.

**Attributes:**

None

**Operations:**

None

**Associations:**

The EnterpriseFramework class has associations with the following classes:

Class: MsAgSentry uses

#### **4.1.3.68 LogFile Class**

Parent Class:Not Applicable  
Public:No  
Distributed Object:No  
Purpose and Description:  
Represents any COTS generated log file

##### **Attributes:**

None

##### **Operations:**

None

##### **Associations:**

The LogFile class has associations with the following classes:

Class: COTSApplication generates  
Class: MsAgEvtLogAdaptor monitors

#### **4.1.3.69 MsAgEvtLogAdaptor Class**

Parent Class:Not Applicable  
Public:No  
Distributed Object:No  
Purpose and Description:

##### **Attributes:**

None

##### **Operations:**

None

##### **Associations:**

The MsAgEvtLogAdaptor class has associations with the following classes:

Class: LogFile monitors  
Class: MsAgSentry uses

#### **4.1.3.70 MsAgSentry Class**

Parent Class:Not Applicable

Public:No

Distributed Object:No

Purpose and Description:

Sentry Monitors system resources and services. It identifies problems before they become critical. In addition, it provides consistency in monitoring remote systems and can be configured to trigger pre-defined automatic actions based on given events.

#### **Attributes:**

None

#### **Operations:**

None

#### **Associations:**

The MsAgSentry class has associations with the following classes:

Class: OperatingSystem ismonitoredby

Class: EnterpriseFramework uses

Class: MsAgEvtLogAdaptor uses

#### **4.1.3.71 OperatingSystem Class**

Parent Class:Not Applicable

Public:No

Distributed Object:No

Purpose and Description:

#### **Attributes:**

None

#### **Operations:**

None

## **Associations:**

The OperatingSystem class has associations with the following classes:  
Class: MsAgSentry ismonitoredby

### **4.1.4 Management Agent Services Dynamic Model**

The management agent services dynamic models are presented in the following subparagraphs. For information on the near real-time Request Tracking capability, refer to Section 6.2.

#### **4.1.4.1 Get MIB Value**

This scenario is depicted in Figure 4.1-15.

##### **4.1.4.1.1 Beginning Assumptions**

The SNMP agent, ECS subagent, and the ECS application are all up and running.

##### **4.1.4.1.2 Interfaces with Other Subsystems and Segments**

Management Framework in MSS Fault Management

##### **4.1.4.1.3 Stimulus**

An MSS management application inquires the value of a MIB variable concerning an ECS application running on a remote host.

##### **4.1.4.1.4 Participating Classes From the Object Model**

EcAgAgent

MsAgSubAgent

EcAgManager

ECS application

##### **4.1.4.1.5 Beginning System, Segment and Subsystem State(s)**

The SNMP agent on the host is up and running. It listens to port 161 to wait for incoming SNMP requests.

The ECS subagent is functioning. It has established connections with the SNMP agent on the same host and waits for incoming requests through the agent-subagent protocol.

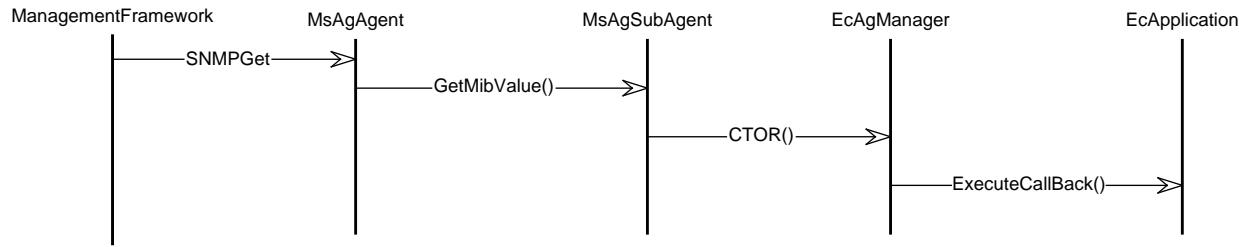
The ECS application is instrumented with the MSS-provided class library which includes the server part of the EcAgManager. This application is also up and running normally. The instrumented EcAgManager object has been instantiated.

##### **4.1.4.1.6 Ending State**

The SNMP agent is still listening to the port 161 for incoming SNMP requests.

The ECS subagent continuously await for incoming requests from the SNMP agent.

The ECS application is running to perform its own functions.



**Figure 4.1-15. Get MIB Value Dynamic Model**

#### 4.1.4.1.7 Scenario Description

A MSS management application needs to check the condition of a managed resource. That can be reflected by the value of a variable defined in ECS application MIB.

The management application issues an SNMP request to retrieve the value of that MIB variable.

The request is passed from MSS Server to the SNMP agent on a particular remote host.

The SNMP agent first does the authentication and authorization validations. If the request is allowed to access that MIB variable, then it checks the MIB registration tree to determine which agent or subagent is responsible for that MIB variable.

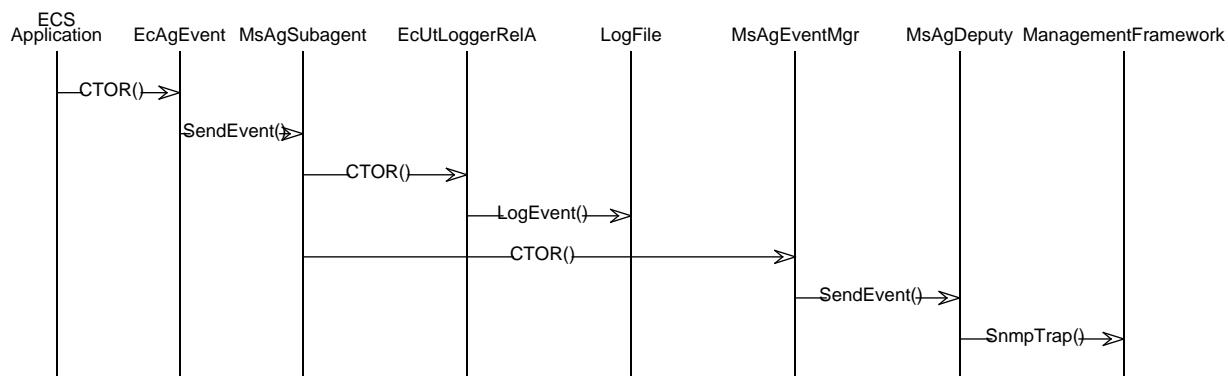
For accessing the application MIB variables, it passes the request to ECS subagent through the agent-subagent protocol which is SNMP MULTipleXing protocol (SMUX).

The subagent is always waiting for incoming requests from the SNMP agent through its MsAgPortMonitor object. When a request comes in, it determines which access method to use for retrieving the requested MIB variable and calls that access method.

The retrieved value will be passed back to SNMP agent and then relayed back to the management application on the MSS Server.

#### 4.1.4.2 SNMP Trap Generation

This scenario is depicted in Figure 4.1-16.



**Figure 4.1-16. SNMP Trap Generation Dynamic Model**

#### **4.1.4.2.1 Beginning Assumptions**

ECS applications are running on a managed ECS host. The SNMP agent and ECS subagent are all up and running. The management framework is running on MSS Server.

#### **4.1.4.2.2 Interfaces with Other Subsystems and Segments**

Management Framework in MSS Fault Management

EcUtLoggerRelA in CSS

#### **4.1.4.2.3 Stimulus**

An error condition occurs in an ECS application. The error is serious enough to inform the Fault Management application which runs on MSS Server.

#### **4.1.4.2.4 Participating Classes From the Object Model**

ECS application

EcAgEvent

MsAgMonitor

MsAgSubAgent

MsAgDeputy

#### **4.1.4.2.5 Beginning System, Segment and Subsystem State(s)**

The ECS application is instrumented with the MSS-provided class library. When the fault condition occurs, ECS application is able to send out event notifications through the sendEvent method of EcAgEvent object. The ECS subagent is up and running on the host. It is listening to receive DCE remote procedure calls.

The Deputy of SNMP manager (management framework) on MSS Server is up and running which is ready to receive DCE remote procedure calls.

#### **4.1.4.2.6 Ending State**

The ECS application may or may not be running caused by the error condition.

The ECS subagent continuous listening to receive DCE RPC calls.

The Deputy of SNMP manager continue to listen to receive DCE RPC calls.

An SNMP trap is generated to the management framework on MSS server. The fault management application detects the fault condition of that ECS application on that host.

#### **4.1.4.2.7 Scenario Description**

When a fault condition occurs in an ECS application, the application instantiates an object EcAgEvent.

The application invokes the sendEvent method on that object sending an event notification to MsAgSubAgent.

The MsAgSubAgent will log the event to MSS log which is managed by CSS EcUtLoggerRelA.

The MsAgSubAgent will check the severity of the event. If it is higher than the infoLevel, then this event notification will go further to the MSS Server.

The MsAgSubAgent will instantiate an object EcAgEvent and invoke its sendEvent method to send this event to the MsAgEventMgr which in turn calls its ProcessEvent operation to send that event to MsAgDeputy on the MSS Server.

The MsAgDeputy will then convert the event to an SNMP trap and send it to the management framework (HP OpenView) by invoking operations SendEvents which in turn calls the SendFaultTrap operation.

#### **4.1.4.3 Sub Agent Startup**

This scenario is depicted in Figure 4.1-17.

##### **4.1.4.3.1 Beginning Assumptions**

The SNMP master agent (MsAgAgent) and the deputy (MsAgDeputy) are already up and running. The management framework is running on the MSS server.

##### **4.1.4.3.2 Interfaces with Other Subsystems and Segments**

Management framework in MSS Fault Management.

##### **4.1.4.3.3 Stimulus**

The subagent has been started and it wants to discover all installed and running applications.

##### **4.1.4.3.4 Participating Classes From the Object Model**

MsAgDiscoverer

MsAgEventMgr

EcAgConfig

EcAgEvent

MsAgRegistry

MsAgTeVerExec

EcAgManager

MsAgDeputyGate

MsagDeputy

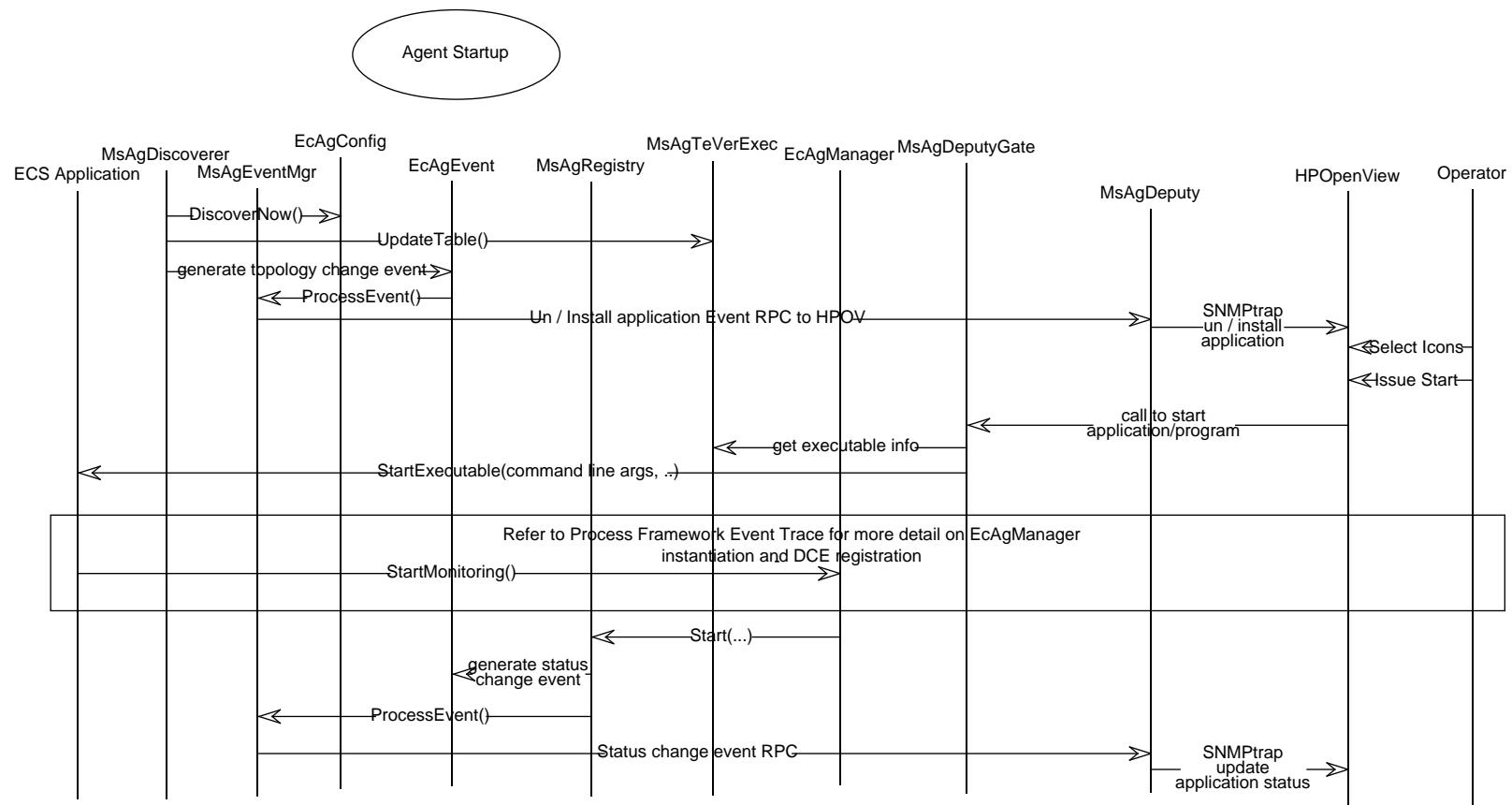
ECS Application

##### **4.1.4.2.5 Beginning System, Segment and Subsystem State(s)**

All the installed applications and executables have their corresponding configuration files in the appropriate sub directory based on the mode. The system has been activated to recognize the desired modes via the ActiveMode operation.

##### **4.1.4.2.6 Ending State**

The application selected by the operator has executed and its status has been updated on the HP Open View map.



**Figure 4.1-17. Subagent startup Dynamic Model**

#### **4.1.4.2.7 Scenario Description**

MsAgDiscoverer issues a DiscoverNow() operation to search for the installed applications and executables based on the active mode list and checks if the installed applications match the last known installed applications as listed in its internal tables MsAgTeStApp and MsAgStProg.

MsAgDiscoverer also writes application related information to subagent's internal table MsAgTeVerExec through AddToTable operation.

If any changes are detected, MsAgDiscoverer generates a topology change event by instantiating an EcAgEvent object and calling its ReportEvent method.

MsAgEventMgr sends the event to MsAgDeputy through ProcessEvent() operation.

MsAgDeputy converts the event to SNMP trap via SendEvents operation and sends it to HP Open View by calling SendTopologyChangeTrap operation. At HP Open View, The application/program is added or removed from the applicable session(s) and submap(s).

At this point the applications appear on a map on HP Open View and can be selected by the operator.

Operator selects the desired application by highlighting its icon and selecting start. Since the icons, which are linked to specific applications/programs, are already tied to a given mode, the operator does not need to enter the mode identifier. The mode has already been established previously.

HP Open View sends an SNMP call to MsAgDeputyGate to start the application.

MsAgDeputyGate gets the executable information from the subagent table MsAgTeVerExec which was written at time of application discovery through AccessTbl operation.

MsAgDeputyGate starts the application by providing the appropriate information (such as the name of the executable and the command line arguments) as arguments to the StartExecutable operation.

EcAgManager registers the metric through RegisterMetric operation and calls the StartMonitoring operation to monitor the application.

MsAgRegistry generates a status change event by instantiating an EcAgEvent object. It then sends the EcAgEvent object to MsAgEventMgr by calling ReportEvent operation, where the event is processed via ProcessEvent operation.

MsAgEventMgr sends the event to MsAgDeputy in the form of a RPC through ProcessEvent operation.

MsAgDeputy converts the event to SNMP trap and sends it to HP Open View through SendEvents operation which in turn calls the SendTopologyChangeTrap operation.

HP Open View changes the status of the application or the program on the map to active.

#### **4.1.5 Management Agent Services Structure**

From a high-level point of view, the Management Agent Service includes five major parts which can be further decomposed as follows:

1. Agent
  - Extensible master agent (SNMP Agent)

- SNMP Subagent(s)
  - Proxy Subagent(s)
2. Monitor
  3. Event Handler
  4. Agent External Interfaces
    - Manager-to-Agent interface
      - Standard management protocol (SNMP)
      - DCE RPC
    - Instrumentation API
  5. Managed Object Model
    - Standard Management Information Base (MIB)
    - Private MIB extensions.

The Management Agent Service can be packaged as shown in Table 4.1-1

**Table 4.1-1. Management Agent Services Components**

Component Name	COTS/Custom
MsAgAgent	COTS
MsAgSubAgent	C/C++ code and COTS
EcAgProxy	C/C++ code
MsAgEncps	COTS
MsAgDeputy	C/C++ code
EcAgManager	C++ code
MsAgAppMib	ASN.1
MsAgSentry	COTS

#### **4.1.5.1 MsAgAgent - Extensible SNMP Master Agent CSC**

##### **Purpose and Description**

In order to manage ECS applications, MSS requires the SNMP agent on each managed host be extensible. The extensible SNMP master agent receives SNMP requests from SNMP managers (management applications) which are on the MSS Server. The agent performs the tasks requested by the management applications. It accesses the managed resources to retrieve MIB values which are supported by this agent. For the MIB objects in extended MIBs, it passes the requests to the subagent(s) to perform the tasks.

The SNMP master agent chosen is Peer Network's agent, along with its toolkit to develop subagents and proxy agents. MSS has to customize the configuration file of the agent.

##### **Mapping to objects implemented by this component**

MsAgAgent - COTS

#### **4.1.5.2 MsAgSubAgent - ECS Subagent CSC**

##### **Purpose and Description**

This subagent supports the Application MIB which is defined by MSS. The functions it supports include:

- Retrieving Application MIB values for management applications.
- Performing local monitoring on applications to avoid excessive remote polling from management applications.
- Sending event notifications to the “deputy” of SNMP manager on MSS Server. That deputy will convert the events to SNMP traps to HP OpenView.
- Logging events onto the local MSS log.

The COTS provides libraries of API for agent-subagent interface and the development environment. The subagent code will be developed by MSS.

##### **Mapping to objects implemented by this component**

###### **MsAgSubAgent**

MsAgSubAgent	- C++ code
MsAgMonitor	- C++ code
MsAgDeputyGate	- C++ code
MsAgDiscoverer	- C++ code
MsAgRegistry	- C++ code
MsAgEventMgr	- C++ code
MsAgEventEntry	- C++ code
MsAgStaticBuffer	- C++ code
MsAgSubAgentConfig	- C++ code
MsAgPortMonitor	- C++ code, PEER toolkit
MsAgCfgFileInfo	- C++ code
MsAgPathFinder	- C++ code
MsAgScheduler	- C++ code
MsAgScheduleEntry	- C++ code
MsAgMgmtBindingVector	- C++ code
MsAgMgmtBindingHandle	- C++ code
MsAgMgmtHandle	- C++ code
MsAgTblMgr	- C++ code
MsAgSNMPTbl	- C++ code
MsAgTblEntry	- C++ code
MsAgTeIndx	- C++ code
- MsagTePerf	- C++ code
- MsAgTeFault	- C++ code

- MsAgTeConfig	- C++ code
MsAgTeMdMg	- C++ code
- MsAgTeVerExec	- C++ code
• MsAgTeStApp	- C++ code
• MsAgTeStProg	- C++ code
- MsAgTeDynamic	- C++ code
• MsAgTeDyProc	- C++ code

#### 4.1.5.3 EcAgProxy - DCE Proxy Agent CSC

##### Purpose and Description

This proxy agent is to provide the capability to manage devices or applications which don't support ECS protocols, namely non-OODCE COTS.

This proxy agent is composed of two parts. Its front-end is the instrumentation code which can communicate with the ECS subagent. Its back-end code uses COTS-specific mechanism to communicate with the COTS.

This code will be developed by MSS and COTS providers. MSS will provide the instrumentation code. The COTS-specific code has to be developed by application developers.

##### Mapping to objects implemented by this component

EsAgProxy	C++ code
EcAgCOTSMgr	C++ code
EcAgCOTSMgrFactory	C++ code
EcAgCOTSLLog	C++ code
EcAgPatternVec	C++ code
EcAgErrPattern	C++ code

#### 4.1.5.4 MsAgEncps - Encapsulator for non-Peer Agent CSC

##### Purpose and Description

This Encapsulator is to enable the Peer master agent to communicate with non-Peer SNMP agent(s).

Some SNMP agents may not be extensible. In order for MSS to manage ECS applications, it is required to have an extensible SNMP agent. Therefore, the Peer's master agent will be used on each ECS managed host. This encapsulator can be used for Peer's master agent to communicate with the original SNMP agent so that the MIB it supports is still accessible.

This is COTS code. MSS only has to provide the customized configuration file.

##### Mapping to objects implemented by this component

- MsAgEncps - COTS

#### **4.1.5.5 MsAgDpty - SNMP Manager's Deputy CSC**

##### **Purpose and Description**

Since SNMP Set requests are not allowed. The Set functions will be done by DCE RPC calls through a “deputy” of SNMP manager on MSS Server. The SNMP manager (HPOV) sends the Set request to this deputy. It will pass the request to the remote application subagent. That subagent will decide what to do next. If it's necessary, it may invoke a call to the ECS application to perform the Set function.

The other function of this Deputy is to receive event notifications in a secure way through DCE RPC from remote hosts. It will convert the event notification to an SNMP trap and send it locally to HP OpenView.

This is a piece of custom code and will be developed by MSS.

##### **Mapping to objects implemented by this component**

MsAgDpty

    MsAgDeputy                  - C/C++ code

    MsagSNMPPdu

#### **4.1.5.6 EcAgInstrm - Instrumentation Class Library CSC**

##### **Purpose and Description**

This class library is provided by MSS to enable the manageability of ECS applications. ECS application developers have to link this class library with their applications so that these applications can communicate with the application subagent.

This instrumentation code will be included by ECS applications and the proxy agent for COTS. It is custom code and will be developed by MSS.

##### **Mapping to objects implemented by this component**

    EcAgManager                  - C++ code

    EcAgShutdown                  - C++ code

    EcAgNamedList                  - C++ code

    EcAgConfigFile                  - C++ code

    EcAgException                  - C++ code

    EcAgMetric                  - C++ code

    EcAgFaultMetric                  - C++ code

    EcAgConfigMetric                  - C++ code

    EcAgPerfMetric                  - C++ code

    MsAgIntConfigMetric                  - C++ code

    MsAgUpDateConfigMetric                  - C++ code

    MsAgProcPerfMetric                  - C++ code

    MsAgProcInfo                  - C++ code

MsAgProcSShotInfo	- C++ code
MsAgMetVector	- C++ code
EcAgEvent	- C++ code
EcAgTuple	- C++ code
MsAgPerfEvent	- C++ code
EcAgHostInfo	- C++ code
MsAgEventHandler	- C++ code
MsAgEventHandler	- C++ code

#### **4.1.5.7 MsAgAppMib - Application MIB CSC**

##### **Purpose and Description**

This class library is provided by MSS to enable the manageability of ECS applications. It includes the Agent group, the Application group, the Program group, the Process group, and the Trap group. The MIB variables of each group are in the spreadsheet format in the appendix of this document.

##### **Mapping to objects implemented by this component**

MsAgAppMib - ASN.1

#### **4.1.5.8 MsAgSentry - Enterprise Framework Agent**

##### **Purpose and Description**

The Tivoli Sentry agent is responsible for monitoring operating system level application fault and performance functionality. In addition, it provides a log event adapter which can be configured to periodically examine COTS log files for specific events and then automatically notify Tivoli when certain conditions occur. Both Sentry and the Log Event Adapter are COTS.

##### **Mapping to objects implemented by this component**

MsAgSentry - COTS

MsAgEvtLogAdaptor - COTS

### **4.1.6 Management Agent Services Management and Operation**

#### **4.1.6.1 System Management Strategy**

The programs in Management Agent Service need to be monitored also.

Although the MsAgMonitor can monitor the MsAgSubAgent, if the subagent terminates abnormally, the monitor may not function. Therefore, it is important for fault management applications to poll the agent and subagents periodically to check the state of the subagents.

The potential performance impact will come from the following areas:

- The capability of processing incoming requests in the SNMP master agent
- The capability of processing incoming requests in the ECS subagent
- The capability of processing incoming requests in the Encapsulator
- The capability of processing outgoing event notifications in the ECS subagent

- The capability of processing outgoing event notifications in the Encapsulator
- The capability of processing outgoing traps in the SNMP master agent
- The number of threads that ECS DCE servers can have.

#### **4.1.6.2 Operator Interfaces**

There are no GUIs for Management Agent Service.

#### **4.1.6.3 Reports**

There are no reports generated from Management Agent Service.